MASTER EN SCIENCES INFORMATIQUES

Génération automatique d'interface d'accès à des bases de données

Bock, P.; Delval, M.

Award date:
1982

Awarding institution:
Universite de Namur

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

GENERATION AUTOMATIQUE D'INTERFACE D'ACCES A DES BASES DE DONNEES

septembre 1982

Mémoire présenté par
P. Bock / M. Delval
en vue de l'obtention du titre de Licencié et Maître en Informatique.

RUE GRANDGAGNAGE, 21, B - 5000 NAMUR (BELGIUM)

E R R A T A

page/ligne	erreur	correction
titre	INTERFACE	INTERFACES
1/1	INTRODUTION	INTRODUCTION
4/6	interface	interfaces
6/21	mémoire.	mémoire).
10/24	lequel	laquelle
15/5	chemin d'accès	chemin d'accès vide
30/figure	descritpion	description
40/35	liges	lignes
41/6	pointeur	pointeur sentinelle
43/2	de zone	de la zone
45/13	chargé	chargée
51/15	notionsd'	notions d'
52/16	unitem	un item
52/21	unitemou	un item ou
59	 ITEM	 ITEM
65/7	sous-schémas	sous-schéma
69/6	C = c	C ≠ c
72/24-26	(à rajouter) <texte 1>#PARAMETREn<texte 2> on aura en sortie : <texte 1>valeur<texte 2>	
72/30	VAR	PARAMETRE
76/27	vérification	vérifications
87/4	untype	un type
98/13	Damstadt	Darmstadt

Nous tenons tout d'abord à remercier Monsieur Jean-Luc Hainaut, promoteur de ce mémoire, pour l'intérêt qu'il a porté à ce travail et pour l'aide qu'il nous a constamment prodiguée, aussi bien lors de l'étude que lors de la rédaction de ce mémoire.

Nous exprimons également toute notre gratitude à Monsieur Victor Meurisse pour l'accueil qu'il nous a réservé au sein de son département chez NCR-European Support Center à Bruxelles, ainsi que pour ses conseils et critiques qui nous ont permis de progresser dans ce travail.

Enfin notre reconnaissance s'adresse à tous les membres de l'Institut d'Informatique, à tous les employés de NCR-European Support Center (Bruxelles) et NCR (Londres) qui, d'une manière ou d'une autre, nous ont aidés au cours de cette année.

TABLE DES MATIERES

INTRODUTION

Pour beaucoup de systèmes informatiques, un point crucial est et restera la manipulation des données. Afin de faciliter la gestion de ces données, de nombreux systèmes plus ou moins évolués ont été mis en œuvre. Il s'agit des Systèmes de Gestion de Fichiers (SGF) et des Systèmes de Gestion de Base de Données (SGBD).

Ces dernières années, nous assistons à une prolifération de tels systèmes, ce qui malheureusement tend à diminuer leur intérêt. En effet, chaque système possède son propre modèle de données et son propre modèle d'accès. Les programmes qui utilisent un tel système ne sont pas indépendants de ces modèles, ce qui limite fortement leur portabilité.

Pour répondre à ce problème, les programmes ne devraient plus utiliser les modèles d'un SGBD particulier, mais un modèle plus général, unique et indépendant d'un SGBD. C'est notamment dans ce but que J.L. Hainaut a conçu le Modèle d'Accès Généralisé (MAG). Cependant, ce modèle n'étant pas compatible avec les modèles des SGBD, il est indispensable de disposer d'une interface entre un SGBD et un programme utilisateur se référant au modèle général.

L'objectif de ce mémoire est de faciliter la réalisation de ces interfaces pour le système DBMS-20 * [D1, D2] par une génération automatique de celles-ci.

---

* DBMS-20 est un SGBD tournant sur le système d'exploitation TOPS-20 de la Digital Equipment Corporation et qui suit le rapport CODASYL d'avril 1971 [C1].

La première partie de ce travail est consacrée à la conception et à la mise en œuvre d'un système de génération indépendamment d'un SGBD particulier.

Le premier chapitre vise à donner une idée plus précise de la notion d'interface dans le cadre de ce travail et à décrire le contexte dans lequel les interfaces interviennent.

Une description relativement sommaire du modèle d'accès généralisé sera faite au chapitre deux.

Au troisième chapitre nous montrerons comment les interfaces peuvent être créées automatiquement par un système de génération d'interfaces. Les différentes architectures du système de génération qui ont été envisagées seront discutées dans ce chapitre.

Dans les chapitres quatre et cinq seront exposés les deux composants principaux du système retenu, qui sont d'une part la base de données des schémas et son SGBD et d'autre part le générateur.

La seconde partie du mémoire traitera du cas plus particulier de la génération d'interface appliquée au cas du système DBMS-20.

Au sixième chapitre sera faite une étude comparative entre le MAG et les modèles de DBMS-20.

Le septième chapitre exposera un élément de l'architecture de génération qui est spécifique au SGBD retenu.

Pour conclure ce mémoire, nous donnerons une évaluation de notre réalisation par rapport aux objectifs fixés, de même qu'un certain nombre de perspectives intéressantes pour ce projet.

PREMIERE PARTIE :

LE SYSTEME DE GENERATION

CHAPITRE 1. LES INTERFACES

Dans ce premier chapitre, nous voulons tout d'abord faire percevoir la notion d'interface base de données et son intérêt dans le contexte général d'indépendance des programmes vis-à-vis des systèmes de gestion de bases de données (SGBD) et des systèmes de gestion de fichiers (SGF). Puis, après avoir donné une définition du concept d'interface dans le cadre de ce mémoire, nous exposerons l'intérêt des interface dans le contexte particulier du projet IDML (Interactive Date Manipulation Language) et dans le contexte plus général de conception d'applications bases de données. Enfin nous essayerons de dégager un premier point de vue critique vis-à-vis des interfaces.

1.1. NOTION D'INTERFACE

Un concepteur de logiciels d'application peut voir une base de données avec son SGBD comme une machine base de données M. Cette boîte noire se charge de conserver et de tenir à jour les informations fournies par l'utilisateur et de les lui envoyer s'il les demande. La machine BD et le programme utilisateur doivent donc dialoguer entre eux, ce qui veut dire qu'ils doivent s'entendre sur un modèle commun ainsi que sur un jeu d'échanges possibles d'informations et sur un vocabulaire à employer (fig. 1.1.1.).

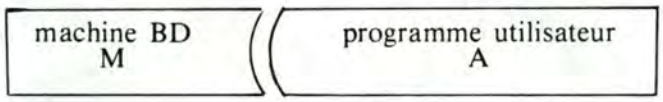


fig. 1.1.1.

Les règles de dialogue sont imposées par le SGBD de la machine BD. Il y a pour une même base de données autant de machines BD possibles qu'il existe de SGBD différents. Cependant, un programme utilisateur A ne pourra dialoguer qu'avec la machine BD pour laquelle il a été conçu, puisque ce sont les seules règles de dialogue qu'il connaît. Si on est cependant amené à changer de machine BD, à cause d'un changement de SGBD, le programme utilisateur ne pourra plus dialoguer avec cette nouvelle machine M' (fig. 1.1.2/a). On se voit contraint à convertir le programme utilisateur A dans un programme A' qui connaît les règles de dialogue observées par la machine M'. (fig. 1.1.2/b).

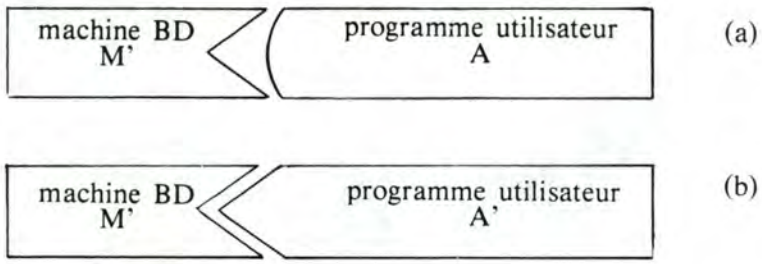


fig. 1.1.2.

On comprend aisément l'hésitation des concepteurs de produits d'application à mettre en œuvre des produits basés sur des bases de données, si l'on considère d'une part que les transformations à apporter aux programmes lors d'un changement de SGBD sont importantes et donc chères, et d'autre part que le risque de changement de SGBD est élevé, vu le fait qu'il existe une grande multiplicité de SGBD sur le marché (IMS, TOTAL, ADABAS, la famille CODASYL et autres) et que cette multiplicité risque de s'accroître suite à l'arrivée de nouveaux SGBD (ADAPLEX, systèmes relationnels...).

Pour être plus souple face à la variété des SGBD, on doit aboutir à une programmation plus indépendante, ce qui veut dire une plus grande indépendance du programme par rapport au modèle de données et aux règles de dialogue de la machine BD [W1] . Ainsi le concepteur d'un programme utilisateur va devoir être à même de construire son programme en ignorant à travers quel SGBD son programme va accéder à la base de données. Les requêtes de l'utilisateur à la machine BD seront formulées dans un langage indépendant du SGBD/SGF en se basant sur un modèle indépendant.

Bien sûr, en général, il sera impossible à ce programme utilisateur de dialoguer directement avec une machine BD car les deux langages sont incompatibles (fig. 1.1.3/a). Le programme utilisateur transmet alors sa requête à son interprète, qu'on appelle interface BD, qui se charge de traduire la requête et puis de retraduire la réponse de la machine BD (fig. 1.1.3/b). L'avantage de cette solution est que le jour où l'on est amené à changer le SGBD de la machine BD, il suffit de changer l'interface, le programme utilisateur pouvant continuer à fonctionner sans qu'on y apporte une quelconque modification (fig. 1.1.3/c).

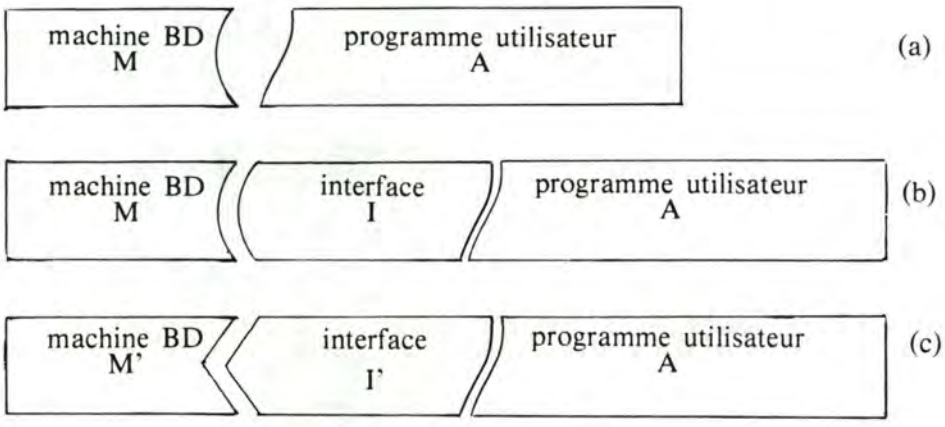


fig. 1.1.3.

L'interface BD, si elle est créée dans l'optique d'un seul programme utilisateur, est telle qu'elle ne permet que la traduction des requêtes formulées par ce programme, ce qui n'est en général qu'un sous-ensemble des requêtes possibles.

Il serait alors intéressant de mettre au point une interface générale pour la machine BD. Dans ce cas, on est amené à considérer que l'interface fait partie intégrante de la machine BD.

Si l'interface doit pouvoir dialoguer avec un ou plusieurs programmes utilisateurs, il faut qu'il y ait une vue commune de la base de données et une entente sur les règles de dialogue. Pour formuler le point de vue commun, nous faisons appel au Modèle d'Accès Généralisé (MAG) élaboré par Monsieur Jean-Luc Hainaut. Le MAG permet d'exprimer la vue commune dans un modèle commun et fournit un jeu de primitives communes et un langage qui met en œuvre ces primitives (ce modèle fera l'objet du 2^{e} chapitre de ce mémoire.

Le choix de MAG comme modèle commun est dû à plusieurs raisons :

- le MAG est un modèle assez général et on peut donc espérer que ce modèle soit assez «résistant» face à des changements de SGBD;
- ce travail s'inscrit dans le contexte des travaux entrepris par l'Institut d'Informatique, dont IDML, qui se basent sur MAG;

- l'élaboration d'un modèle cohérent et général est un travail important, qui ne fait pas partie du sujet proprement dit de ce mémoire.

1.2. DEFINITION

Après avoir découvert l'intérêt d'une interface, il faut se demander quel niveau de généralité il convient de lui donner. Faut-il avoir :

- une interface pour toute BD sur tout SGBD, ou
- une interface pour toute BD sur un SGBD, ou
- une interface pour une BD sur un SGBD, ou
- une interface pour un sous-schéma sur un SGBD ?

Pour pouvoir trancher, voyons ce que ces différents choix impliquent.

Une interface pour toute BD sur tout SGBD doit être capable, lorsqu'elle reçoit une requête d'un utilisateur, de construire dynamiquement la requête vers la machine BD car, en effet, dans l'interface, les traductions des requêtes ne peuvent pas être prévues statiquement comme elles sont dans un nombre illimité. Pour pouvoir traduire la requête, l'interface doit alors consulter d'une part la description du SGBD et d'autre part la description de la BD. Le même problème se pose lorsqu'il faut retraduire la réponse de la machine BD. Le processus de consultation à chaque requête risque fort de diminuer la performance de l'interface. D'autre part, certains SGBD n'acceptent pas ou difficilement l'appel par paramètres, ce qui exclut cette solution pour ces SGBD.

La solution d'avoir une interface par SGBD permet d'augmenter les performances, puisqu'il ne faut plus consulter la description du SGBD, mais reste néanmoins le problème qu'il faut pouvoir construire dynamiquement les appels, donc se pose toujours le problème des paramètres.

Cependant, si l'on a une interface par BD pour un SGBD, toutes les traductions des types de requêtes sont prévisibles et sont dans un nombre limité. On peut donc les intégrer directement dans l'interface et celle-ci ne doit plus consulter les descriptions de la BD ou du SGBD. En outre, le problème des paramètres ne se pose plus. Mais il faudrait que l'interface puisse vérifier que la reqûete qui lui parvient est acceptable dans le cadre du sous-schéma auquel l'utilisateur s'adresse. Pour faire cette vérification, l'interface doit pouvoir consulter la description des sous-schémas.

Comme cette consultation risque à nouveau de diminuer les performances, on est amené à proposer d'avoir une interface pour un sous-schéma sur un SGBD.

Pour la suite de ce travail, nous définissons [R1] une interface comme un programme spécifique à un sous-schéma de base de données sur un SGBD, chargé de réaliser toutes les requêtes possibles émises par des utilisateurs et qui concernent cette base de données. Une requête consiste en une demande d'exécution d'une des primitives définies dans le modèle d'accès généralisé.

Cependant, si l'on veut avoir une interface pour chaque sous-schéma, il faut la créer, ce qui impose tout de même un travail important. Vu cette limitation, on ne peut raisonnablement envisager une solution par interface que si l'on parvient à les créer d'une façon assez aisée. Dans la suite, nous proposerons une solution qui permet de créer automatiquement ces interfaces.

1.3. LE CONTEXTE IDML

Le but principal du projet IDML (Interactive Data Manipulation Language) [H2, H4, H5, D3] est la réalisation d'un système interactif d'exploitation de base de données qui se distingue d'autres systèmes par plusieurs caractéristiques.

D'abord, l'accès aux bases de données est offert à la fois à l'usager occasionnel ainsi qu'à un programmeur d'application. Ensuite,le système n'exige pas des bases de données qui lui soient adaptées, mais il accepte toute base de données existante et permet la coexistence de bases de données hétérogènes, c'est-à-dire gérées par des SGBD ou des SGF (Systèmes de Gestion de Fichiers) différents. Enfin, le système doit être portable vis-à-vis des machines et des SGBD/SGF et il doit être extensible.

Pour réaliser l'objectif du projet, le système IDML doit assurer le traitement des programmes IDML à leurs différents stades. Le système doit donc assurer le dialogue avec l'utilisateur, gérer l'introduction, la correction et la conservation des programmes, assurer et contrôler l'exécution et la mise au point de ceux-ci, et par conséquent gérer les communications avec les machines BD.

Le système IDML est composé de plusieurs modules réalisant les tâches décrites ci-dessus. Suite à une demande d'un programme IDML qu'ils prennent en charge, certains des modules du système sont amenés à formuler des requêtes à une des machines BD sur lesquelles le système IDML travaille. Or, les machines BD ne sont pas toujours les mêmes, car d'une part le SGBD/SGF peut changer et d'autre part, on peut ajouter ou retirer des machines BD du système. Mais il faut que le système IDML reste invariant vis-à-vis de ces changements. Pour réaliser cete invariance, il faut que toutes les machines BD que l'utilisateur veut faire ac-

cepter par le système IDML connaissent les règles de dialogue du système IDML. Ces règles sont les règles définies par le modèle d'accès généralisé (cfr chapitre 2).

En général une machine BD composée de la base de données et d'un SGBD/SGF ne connaît pas ces règles. Il faut donc ajouter à cette machine initiale une interface BD qui soit capable de traduire les requêtes exprimées par le système selon les règles du modèle d'accès généralisé, dans une ou plusieurs requêtes compréhensibles par le SGBD/SGF (fig. 1.3.1.).

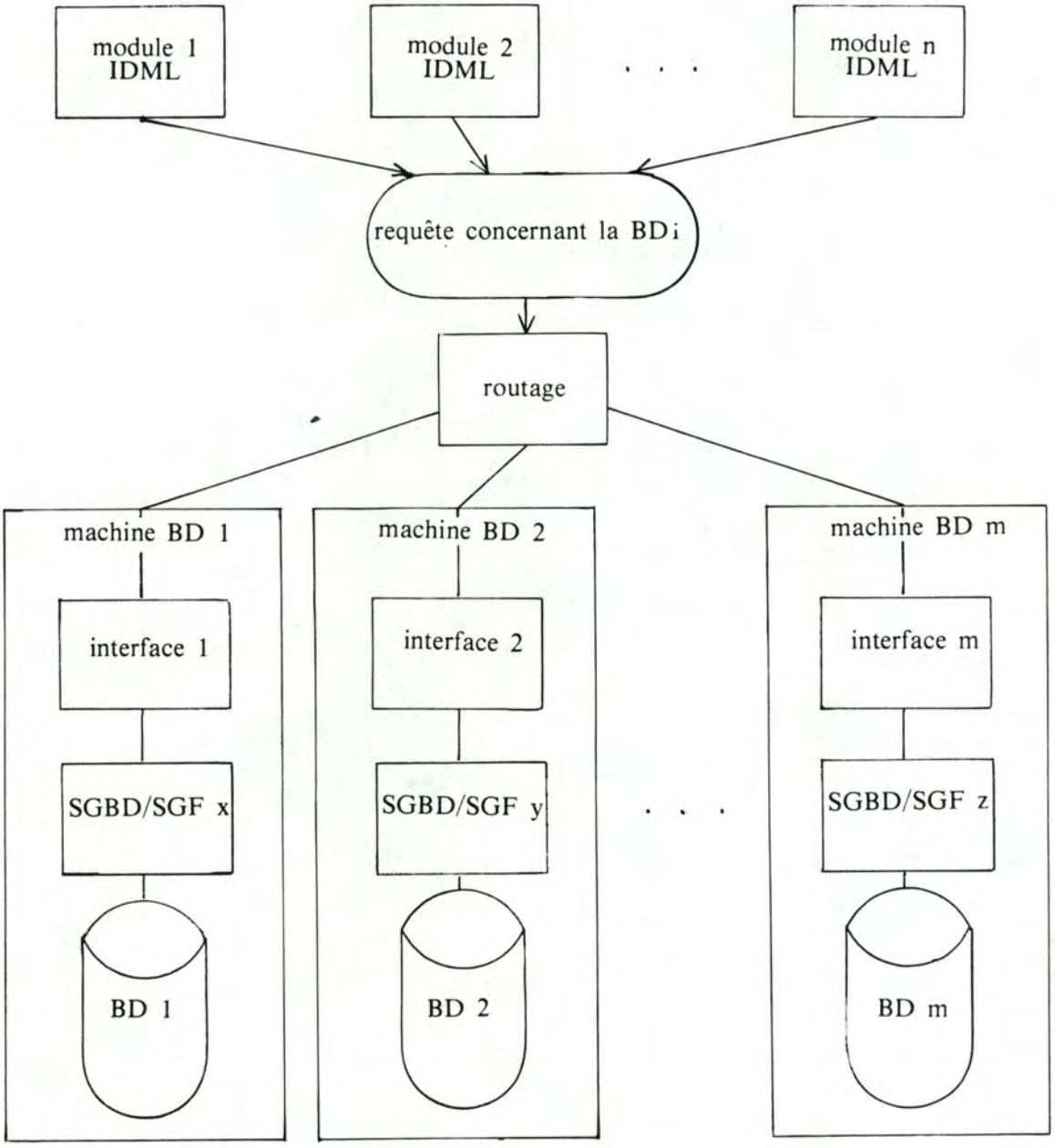


fig. 1.3.1

Evaluons maintenant la solution adoptée par rapport aux objectifs énoncés pour le projet.

Le système actuel permet de reprendre des bases de données existantes et non propres au système, à condition d'ajouter à la machine BD une interface adéquate.

Le système accepte aussi la coexistence de plusieurs SGBD/SGF car l'existence de ceux-ci est transparente pour lui. Pour la même raison, le système est protable vis-à-vis des SGBD/SGF.

Les autres objectifs, tels la portabilité vis-à-vis du système d'exploitation et le fait d'accepter des utilisateurs occasionnels et des programmes, sont réalisés dans les parties du système IDML qui ne sont pas vues ici.

Ainsi donc, la partie du système IDML présentée ici réalise bien les objectifs qu'elle peut prendre en charge, à condition qu'on soit capable de se munir assez facilement des interfaces-BD nécessaires.

1.4. LE CONTEXTE GENERAL

Au-delà du contexte IDML, l'approche par interfaces peut être intéressante chaque fois qu'on veut concevoir une application faisant appel à des bases de données.

Les concepteurs de logiciels d'application peuvent espérer que, grâce aux interfaces, la façon de voir une base de données reste stable du point de vue des structures de données et du point de vue des primitives et du langage. Ceci a l'avantage que lors d'un changement de SGBD, il ne faut pas changer les applications et il ne faut pas introduire tous les programmeurs dans la façon de voir les choses sur lequel est basé le SGBD. Cette indépendance permettra plus facilement de changer de SGBD car, pour pouvoir continuer à travailler, il suffira de se munir des interfaces adéquates.

1.5. PREMIER POINT DE VUE CRITIQUE

Un problème que nous n'avons pas soulevé jusqu'ici est le fait qu'un modèle commun, que ce soit le modèle d'accès généralisé ou un autre, ne fournit en général pas les mêmes possibilités au niveau des structures de données et des primitives qu'un SGBD donné. Est-ce que l'interface peut se contenter de ne four-

nir un service que pour l'intersection des possibilités du modèle général d'un côté et des possibilités du SGBD de l'autre ?

Si le SGBD dispose d'une facilité dont ne dispose pas le modèle général, on peut s'interdire d'utiliser cette possibilité. Mais ceci peut limiter la reprise des bases de données préexistantes.

Si le modèle commun dispose d'une facilité dont ne dispose pas le SGBD, on peut aussi s'interdire de l'utiliser. Mais alors, si l'on passe d'un SGBD disposant de la possibilité à un SGBD qui ne l'a pas, les programmes d'application qui l'utilisent doivent être modifiés.

On voit donc que si l'on ne veut pas restreindre l'intérêt des interfaces, il faudra veiller à ce que les interfaces émulent des structures de données et des primitives. Pour pouvoir émuler, il faut savoir comment on est passé du schéma lié aux SGBD au schéma lié au modèle général, ou vice versa. Ainsi, au moment où l'on crée les interfaces, que ce soit automatiquement ou non, il faut disposer d'une certaine information pour pouvoir faire l'émulation. Il faut, à chaque fois qu'on veut intégrer un nouveau SGBD au système, élaborer des règles qui déterminent les passages possibles et les passages nécessaires d'un sous-schéma compatible avec le modèle du SGBD à un sous–schéma compatible avec le modèle en général. En plus on doit déterminer quelles sont les informations nécessaires pour pouvoir faire les émulations qui s'imposent.

CHAPITRE 2. LE MODELE D'ACCES GENERALISE

Après avoir relevé l'intérêt du modèle d'accès généralisé, nous exposerons ce modèle d'une façon succinte au niveau des structures de données et au niveau des primitives. Le lecteur trouvera plus de détails dans le « Manuel d'utilisation des interfaces » (Annexe). Si le lecteur s'intéresse plus en détail à ce modèle, nous le renvoyons aux références [H1, H6] sur lesquelles nous nous sommes basés.

2.1. INTRODUCTION

Le Modèle d'Accès Généralisé (MAG) permet de décrire les structures de données et les caractéristiques d'accès aux données, comme le permettent de décrire d'autres SGBD.

Il faut remarquer que ce modèle n'est pas un modèle conceptuel car il décrit des propriétés d'accès, il admet de la redondance dans les données et on peut inclure dans un schéma d'accès des considérations d'efficacité. D'autre part, si le modèle décrit les possibilités d'accès, il ne décrit nullement l'implémentation et il ne se situe donc pas non plus à un niveau physique. Il faut situer ce modèle à un niveau intermédiaire appelé le niveau logique. Ce niveau logique est le niveau connu par les programmes utilisateurs.

Ce modèle d'accès généralisé a été conçu par Jean-Luc Hainaut à l'Institut d'Informatique à Namur pour être un outil puissant dans différents champs d'utilisation.

Le modèle permet d'enseigner les concepts fondamentaux de fichiers et bases de données sans se référencer à un SGBD particulier.

On peut décrire les concepts de chaque SGBD dans les termes du MAG, ce qui permet une comparaison facile entre différents SGBD.

Ce modèle permet de construire un schéma logique d'accès qui est encore indépendant d'un SGBD donné.

En plus, on voudrait mettre à disposition des programmeurs ce point de vue indépendant du SGBD, afin de leur cacher le fait que leurs programmes font appel à un SGBD particulier avec des caractéristiques bien particulières et par là-même rendre les programmes indépendants vis-à-vis des SGBD utilisés.

Comme ce dernier objectif du modèle d'accès généralisé rencontre bien le but de notre travail, on peut comprendre aisément notre choix du MAG comme modèle général et unique pour les bases de données.

2.2. LES OBJETS DU MODELE D'ACCES GENERALISE

2.2.1. L'article

L'article est une unité d'information enregistrée qui peut faire l'objet d'un accès, d'une création, d'une modification ou d'une suppression. Chaque article peut être identifié parmi tous les autres articles de la base de données.

L'article constitue l'unité logique de transfert entre la base de données et le programme d'application.

2.2.2. Le type d'article

Chaque article d'une base de données appartient à un, et à un seul type d'article qui définit les propriétés générales de tous les articles de ce type. Dans une base de données, un type d'article est identifié par son nom.

2.2.3. La valeur d'item

La valeur d'item est une donnée appartenant à un type de données (appelé domaine) et qui peut être manipulée par un programme.

2.2.4. L'item

Les propriétés des valeurs d'item sont décrites par les items auxquels elles sont associées. L'item est un type d'information défini par un ensemble de valeurs (= domaine). Un item est associé à au moins un type d'article. Il a un nom qui l'identifie parmi tous les items rattachés au même type d'article. Deux items sont comparables si leurs domaines sont comparables.

Un item peut être élémentaire ou décomposable. Du point de vue de sa signification, la valeur d'un item élémentaire est atomique. La valeur d'un item décomposable est composée de valeurs appartenant à des items simples et/ou décomposables.

Un item peut être simple ou répétitif. On dira qu'un item est simple si à chaque article n'est jamais associée plus d'une valeur de cet item, alors qu'on dira qu'il est répétitif dans le cas contraire.

Un item peut être obligatoire ou facultatif. Un item est dit obligatoire pour un type d'article si à chaque article de ce type est associé au moins une valeur de cet item. Un item est dit facultatif s'il est permis de ne pas associer une valeur de cet item à l'article du type donné.

2.2.5. L'article système

Toute base de données contient un et seul article d'une type particulier, appelé article système.

Cet article constitue un point d'entrée privilégié dans la base de données. Il peut être origine de chemin d'accès, mais non pas une cible.

Comme l'existence de cet article système est liée à celle de la base de données, il portera en général le nom de la base de données.

2.2.6. Le fichier

Un fichier est une collection dynamique d'articles. Chaque article appartient toujours à un et un seul fichier. Un fichier peut contenir des articles de types différents et des articles d'un même type peuvent être enregistrés dans des fichiers différents.

Un fichier peut appartenir à plusieurs bases de données. Les fichiers d'une base de données sont identifiés par leur nom.

2.2.7. La base de données

Une base de données est la collection des articles d'un ensemble de fichiers et de toutes les structures qui ont été définies sur ces articles.

2.2.8. Le chemin d'accès

Un chemin d'accès entre articles est un mécanisme qui associe un article, dit origine, à zéro, un ou plusieurs articles dits cibles, et ceci d'une façon telle qu'ils soit possible d'accéder successivement, à partir de l'article origine, aux différents articles cibles associés.

Un chemin d'accès contient un article origine mais aucun article cible.

2.2.9. Le type de chemin d'accès

Tout chemin d'accès appartient à un et un seul type de chemin d'accès qui définit les propriétés communes de tous ses chemins. Un type de chemin d'accès est caractérisé entre autres par les types d'articles auxquels doivent appartenir d'une part les articles origines et, d'autre part, les articles cibles.

A tout moment, il y autant de chemins d'accès d'un type donné qu'il y a d'articles des types origines.

Un chemin d'accès est caractérisé aussi par les contraintes sur les modes d'insertion d'un article dans un chemin et sur les possibilités de retrait d'un article d'un chemin.

2.2.10. La clé d'accès

Une clé d'accès est caractérisée par le référentiel dans lequel elle est définie. On retient les référentiels suivants :

- tous les articles de la base de données
- tous les articles d'un type dans la base de données
- tous les articles d'un fichier
- tous les articles d'un type dans un fichier
- tous les articles d'un chemin d'accès
- tous les articles d'un type dans un chemin d'accès.

Une clé d'accès est un item ou un groupe d'items d'un type d'article tel qu'il existe un mécanisme qui permet d'accéder successivement aux articles du référentiel donné, auxquels est associée une valeur donnée pour ce groupe d'items.

2.2.11. L'identifiant

Un identifiant est défini dans un référentiel donné. Les référentiels possibles sont les mêmes que ceux retenus pour la clé d'accès.

Un identifiant est un groupe d'un ou plusieurs items qui vérifie d'une part qu'à chaque article du référentiel est associé un tel groupe et, d'autre part, qu'il n'existe pas deux articles du référentiel qui aient même valeur pour ce groupe.

2.2.12. L'ordre

La plupart des primitives d'accès aux données offrent un accès séquentiel aux articles d'un référentiel donné dans la base de données, selon un certain ordre.

On distingue quatre classes principales de primitives selon le référentiel auquel elles se rapportent :

- l'accès séquentiel aux articles de la base de données
- l'accès séquentiel aux articles d'un fichier
- l'accès séquentiel aux cibles d'un chemin d'accès
- l'accès séquentiel aux articles qui ont une certaine valeur pour une clé d'accès donnée.

Il faut envisager des ordres différents selon qu'il y a un seul type d'article ou plusieurs types d'articles dans le référentiel.

Si le référentiel contient un seul type d'article, on retiendra les quatre types d'ordre suivants :

- ordre non-significatif
- ordre lié au moment d'insertion : chronologique ou antichronologique
- ordre trié selon les valeurs d'une clé de tri : croissante ou décroissante
- ordre programmé : la position d'un article est choisi par le programme après ou avant un certain article.

Si le référentiel contient au moins deux types d'articles différents, on retiendra les types d'ordre suivants :

- tout le référentiel est ordonné selon un des quatre types définis ci-dessus, sans tenir compte des types d'articles
- un tri majeur est maintenu sur le type d'article, de façon à ce que tous les articles d'un type soient regroupés; à l'intérieur de chaque type d'article est appliqué un des quatre types d'ordre définis ci-dessus;
- aucun ordre n'est maintenu entre des articles de différents types, mais pour les articles d'un même type, on applique un des quatre types d'ordre définis ci-dessus.

2.2.13. La référence

Une valeur spéciale, contrôlée de manière interne, est associée à chaque article de la base de données; bien que n'étant pas véritablement un item, ces valeurs peuvent être considérées comme constituant un identifiant dans le référentiel : tous les articles de la base de données.

2.3. LES PRIMITIVES DU MODELE D'ACCES GENERALISE

Les primitives sont les opérations qu'un programme peut réaliser sur les données décrites dans la partie précédente.

Il faut remarqer que les primitives décrites par le modèle d'accès généralisé ne constituent pas un langage de manipulation de données. Le langage par lequel ces primitives ont été mises en œuvre pour l'utilisation des interfaces est décrit dans le «Manuel d'utilisation des interfaces» (Annexe).

On distingue trois classes de primitives :

- les primitives d'accès
- les primitives de mise à jour
- les primitives de contrôle.

2.3.1. Les primitives d'accès

Les principaux objets à accéder sont la base de données, le fichier, l'article et la valeur d'item.

2.3.1.1. – Accès à la base de données

- Ouverture de la base de données :

  permet de rattacher un utilisateur (programme) à la base de données s'il y est autorisé; l'utilisateur doit préciser le niveau de protection qu'il demande.

- Fermeture de la base de données :

  libère la base de données ainsi que les restrictions éventuelles imposées lors de l'ouverture.

2.3.1.2. – Accès aux fichiers

- Ouverture d'un fichier :

  ceci correspond à l'ouverture classique des fichiers; ici aussi l'utilisateur doit préciser le niveau de protection qu'il demande. Avant d'accéder à un article, le fichier correspondant doit être ouvert.

- Fermeture d'un fichier :

  libère le fichier ainsi que les restrictions éventuelles imposées lors de l'ouverture.

2.3.1.3. – Accès aux articles

Les primitives suivantes permettent d'accéder à des articles et à leurs valeurs d'items. Elles peuvent être rassemblées dans une seule catégorie de primitives que nous appellerons :

«l'accès dans une séquence à un sous-ensemble ordonné d'articles»

Les différents types d'accès repris sous cette catégorie se distinguent par la valeur de deux paramètres :

- la définition du sous-ensemble ordonné
- la position de l'article désiré dans le sous-ensemble.

Les différents sous-ensembles possibles sont :

- les articles de la base de données
- les articles d'un fichier
- les articles cibles d'un chemin d'accès
- les articles associés à une valeur donnée de clé d'accès, dans un référentiel déterminé, qui peut être la base de données, un ou plusieurs fichiers ou un type d'article.

Chacun de ces sous-ensembles peut être réduit par la sélection, à l'intérieur de ceux-ci, des articles d'un type déterminé. Ainsi nous obtenons quatre autres sous-ensemble possibles :

- les articles d'un type de la base de données
- les articles d'un type d'un fichier
- les articles d'un type cible d'un chemin d'accès
- les articles d'un type associés à une valeur donnée de la clé d'accès dans un référentiel déterminé, qui peut être la base de données, un ou plusieurs fichiers ou le type d'article.

Si plus d'un ordre a été déclaré pour ces sous-ensembles, alors un ordre peut être spécifié :

- naturel : il s'agit de l'ordre utilisé par défaut ou imposé, s'il en existe un;
- (anti–) chronologique : - du premier au dernier
  - du dernier au premier
- trié : dans l'ordre des valeurs d'une clé de tri déterminée, cet ordre pouvant être croissant, décroissant, ...

Comme position de l'article désiré, les possibilités sont les suivantes :

- premier, dernier
- suivant, précédent par rapport à un article précédemment accédé
- ième, -ième par rapport à un article précédemment accédé

a partir de ces considérations, on dégage les primitives suivantes :

- Accès aux articles de la base de données :

  étant donnée la référence de la base de données, cette fonction fournit la référence :
  - du premier, dernier article
  - du ième article du début ou de la fin

- de l'article précédent ou suivant par rapport à un article donné de la base de données.

Un type d'article et un ordre peuvent être spécifiés.

- Accès aux articles d'un fichier :

  étant donnée la désignation du fichier, cette fonction renvoie la référence de l'article qui a la position spécifiée (comme pour l'accès aux articles de la BD).
  Un type d'article et un ordre peuvent être spécifiés.

- Accès aux articles cibles d'un chemin d'accès :

  étant donnée la désignation d'un chemin d'accès, cette fonction fournit la référence de l'article dans le chemin d'accès qui occupe la position spécifiée.
  Un type d'article et un ordre peuvent être spécifiés.

- Accès aux articles auxquels est associée une valeur de clé donnée :

  étant données une clé d'accès et une valeur de clé correspondante, cette fonction fournit la référence d'un article ayant la position spécifiée dans la séquence des articles auxquels est associée cette valeur de clé.
  Un type d'article et un ordre peuvent être spécifiés.

- Accès à un article par identifiant interne :

  étant donnée la valeur d'un identifiant interne d'un article, cette fonction renvoie la référence de l'article identifié ainsi.

- Accès aux items d'un article :

  étant données la référence de l'article et une liste de noms d'items, cette fonction renvoie les valeurs de ces items pour l'article référencé.

- Accès au type d'article, au fichier et à l'identifiant interne d'un article :

  étant donnée la référence d'un article, cette fonction fournit les désignations du type d'article, le nom du fichier et la valeur de l'identifiant interne de l'article.

2.3.2. Les primitives de modification

- Création d'un article :

  si les contraintes d'intégrité définies dans le schéma peuvent être vérifiées à l'aide des données en entrée, cette fonction enregistre dans la base de données un article en accord avec le mode de stockage du type d'article spécifié et insère l'article dans les chemins d'accès automatiques.

- Suppression d'un article :

  étant donnée la référence d'un article, cette fonction retire cet article de tous

les chemins d'accès dont il est cible, détruit les chemins d'accès dont il est origine, supprime éventuellement les cibles de ces chemins et enfin retire l'article du fichier.

- Modification des valeurs d'item d'un article :

  étant donnée la référence d'un article, le nom des items et les valeurs d'item, cette fonction ajoute de nouvelles valeurs à l'article, retire des valeurs ou change des valeurs de l'article.

- Insertion d'un article dans un chemin :

  étant données la référence de l'article et la référence du chemin d'accès, cette fonction insère l'article dans le chemin d'accès.
  Un point d'insertion peut être spécifié.

- Retrait d'un article d'un chemin d'accès :

  étant données la référence de l'article et la référence du chemin d'accès, cette fonction retire l'article du chemin d'accès.

- Changement de chemin d'accès :

  étant données la référence d'un article et les références de deux chemins d'accès du même type, cette fonction retire l'article du premier chemin d'accès et l'insère dans le deuxième.

2.3.3. Les primitives de contrôle

Ces primitives doivent assurer un fonctionnement correct entre une base de données et plusieurs utilisateurs. Elles doivent donc apporter des solutions aux problèmes de concurrence, de reprise et de consistance des données par rapport aux contraintes d'intégrité qui ne sont pas directement prises en charge par les SGBD.

- Début d'une séquence atomique :

  marque un moment à partir duquel les modifications apportées à la base de données sont réversibles jusqu'à ce que la primitive de fin de séquence atomique est réalisée.

- Fin d'une séquence atomique :

  les modifications apportées à la base de données depuis le dernier «début d'une séquence atomique» sont considérées comme définitives.

- Echec d'une séquence atomique :

  les modifications apportées à la base de données depuis le dernier «début d'une séquence atomique» sont retirées de la base de données.

A l'aide de ces trois primitives, le programmeur peut concevoir des super-primitives pour faire face aux problèmes cités ci-dessus.

CHAPITRE 3. GENERATION AUTOMATIQUE

3.1. INTRODUCTION

Afin de bien percevoir la notion de génération automatique telle que nous l'entendrons dans la suite de ce travail, nous allons donner un exemple simple qui reprendra tous les éléments de base d'un principe de génération.

Dans la plupart des entreprises, il est souvent nécessaire de rappeler aux clients de payer certaines factures qui sont venues à échéance. Ces rappels ont en général la forme suivante :

Les éditions européennes
Chaussée de Wavre, 114
B-1330 RIXENSART

Jacques Bintz fils senc
Dernier sol 3
2543 LUXEMBOURG

EXTRAIT DE COMPTE

N. Client	N. Ext	Date
10000462	83	09.04.82

Les factures reprises ci-dessous venues à échéance ne sont pas encore payées. (Compte arrêté au 31.03.82). Prière de vérifier et de payer le présent solde.

N. Fac	Date	Date éch.	Montant fac.	Déjà payé	Solde
46295	30.09.81		932	0	932
50212	15.02.82		932	0	932
CCP Brux. : 000-0350972-26				**TOTAL**	1864
CCP Luxembourg : 975-69					

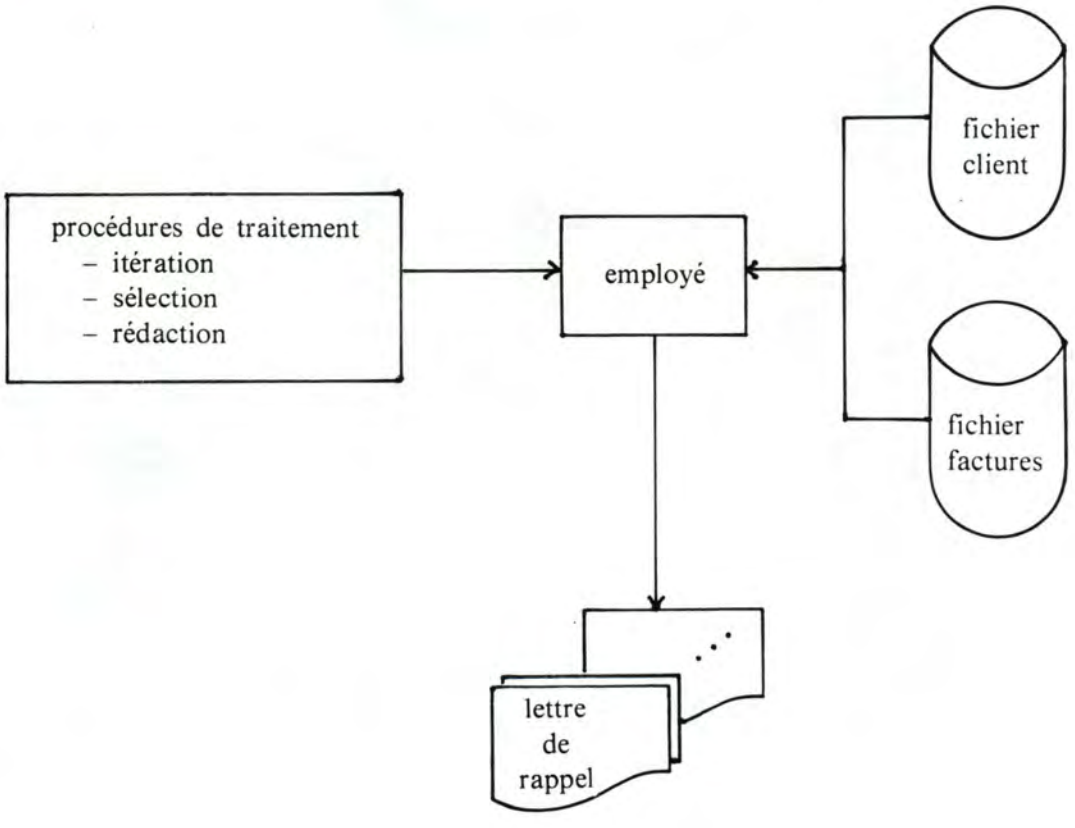
La démarche qui permet l'élaboration de toutes les lettres de rappel à une date précise, que ce soit automatisé ou non, sera la suivante :

Pour chaque client, il faut vérifier s'il n'existe pas des factures venues à échéance et non encore réglées totalement. S'il en existe au moins une, on rédige la lettre de rappel en inscrivant le nom, l'adresse, la référence, ... du client; ensuite, pour chacune des factures non payées et arrivées à échéance, on écrit une ligne comportant le numéro, la date, la date d'échéance, le montant facturé, le montant déjà payé et le montant restant à payer.

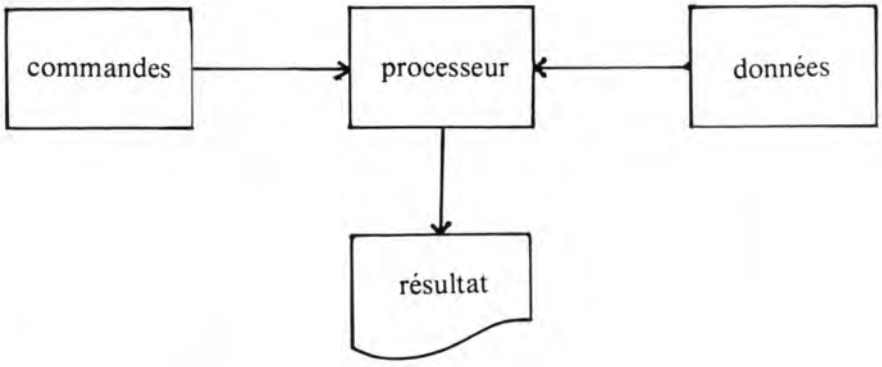
Dans ce processus, il apparaît les structures de traitement suivantes :

- l'itération : «pour chaque client...»
- l'itération imbriquée : «pour chaque client ...»
  «pour chaque facture de ce client ...»
- la sélection : «facture arrivée à échéance et non payée»
- la rédaction : -d'un texte fixe : tout ce qui est en gras
  -d'un texte variable en fonction des clients et des factures.

Ce processus peut finalement être schématisé de la façon suivante :



ou encore en généralisant les concepts :



3.2 ARCHITECTURES

3.2.1. Les différents niveaux d'indépendance

Comme nous l'avons vu dans le § 1.2., les interfaces que l'on désire générer automatiquement ne seront utilisés que pour une base de données bien précise. A partir du moment où il est question de génération «automatique», il est indispensable que celle-ci soit possible quel que soit le schéma de la base de données pour laquelle l'interface doit être généré. Ceci constitue le premier niveau d'indépendance de l'architecture de génération, à savoir l'indépendance par rapport au schéma de B.D.

Jusqu'ici, nous ne nous sommes pas préoccupés de savoir par quel SGBD (Cobol, Codasyl, Total, ...) les bases de données étaient gérées. Il faut noter à ce sujet que toute interface générée sera toujours liée au SGBD qui gérera la B.D. Vu que l'on ne connaît pas à priori quels seront ces SGBD, il serait intéressant de disposer d'une architecture qui permette de générer des interfaces quel que soit le SGBD qui gérera les B.D. Ce problème apparaît clairement dans le projet IDML où les différentes B.D. ne sont pas nécessairement toutes gérées par le même SGBD. Ce deuxième niveau d'indépendance constitue l'indépendance par rapport au SGBD.

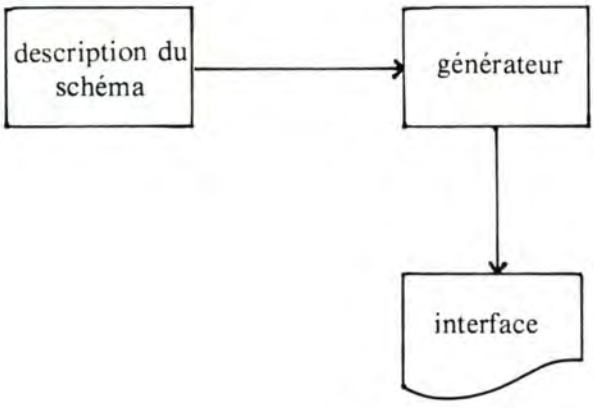
Finalement, on peut encore imaginer un troisième niveau d'indépendance : l'indépendance par rapport au modèle d'accès. Ce troisième niveau dépasse l'objectif de ce travail qui est, rappelons-le, de réaliser un générateur d'interface entre le MAG et un SGBD déterminé. Néanmoins, il pourrait se justifier si, pour l'une ou l'autre raison, on désirait travailler avec un modèle d'accès ayant une structure de données différente de celle du MAG.

Dans les paragraphes qui suivent, nous exposerons, pour chacun des niveaux d'indépendance (schéma, SGBD, modèle d'accès), les différentes architectures de génération que nous avons envisagées, de même que la démarche qui nous a menés à retenir celle qui nous a paru la plus adéquate.

3.2.2. Premier niveau : indépendance par rapport au schéma.

Une interface, dans le sens où nous l'avons définie dans le premier chapitre, peut être considérée comme un objet de génération, c.à.d. qu'elle constitue un texte généré en fonction d'un certain nombre de données. Ces données étant en fait la description du schéma de la B.D.

Le premier niveau d'indépendance sera réalisé en considérant le générateur comme une procédure, appelée avec un certain nombre de données, et ayant pour fonction de produire un texte déterminé en fonction de ces données.



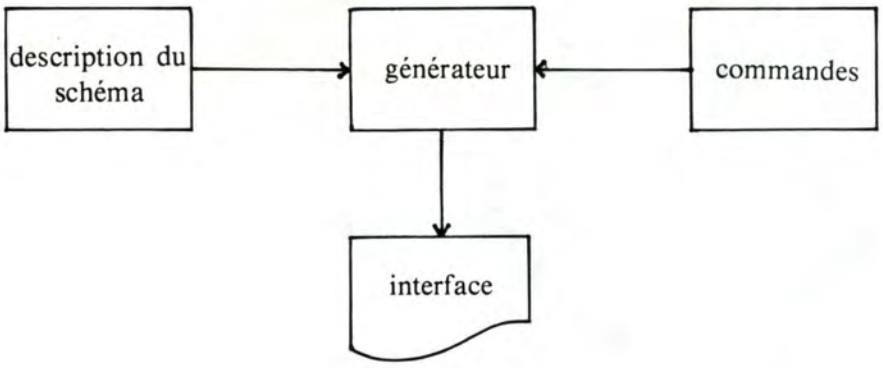
On constate, dans cette architecture, que les éléments d'un système de génération, que nous avons repris dans l'exemple de la lettre de rappel, sont en majeure partie inclus dans le générateur même. Il s'agit des éléments tels que le texte fixe et les commandes.

Il existe de nombreux générateurs travaillant de la sorte, notamment dans le domaine précis de la génération automatique d'interface, nous pouvons citer le package IDM-9000 de NCR.

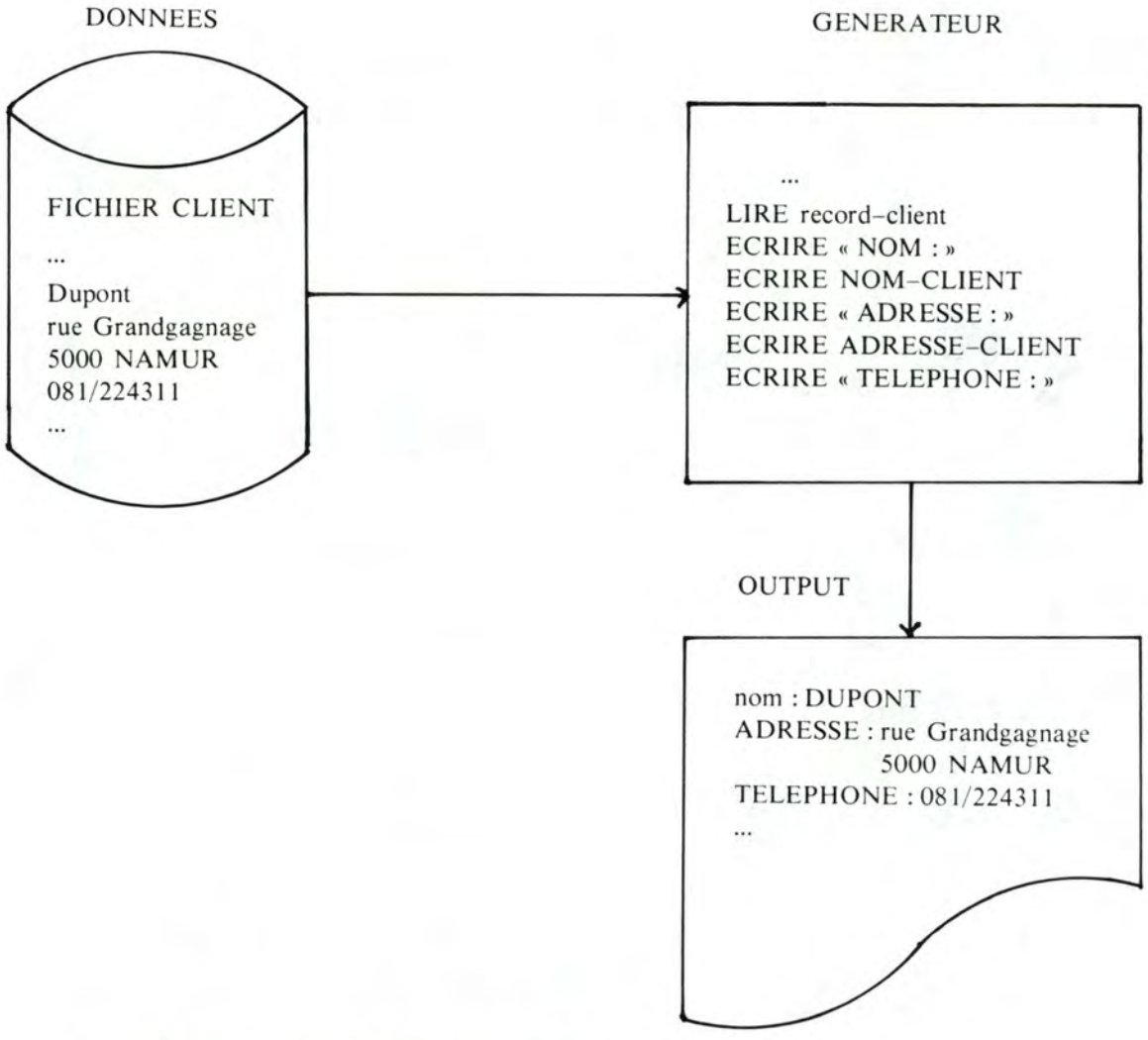
3.2.3. Deuxième niveau : indépendance par rapport au SGBD

La première architecture est sans aucun doute la plus simple, mais ne nous aurait pas permis de réaliser l'objectif de ce travail; c'est pourquoi une architecture tenant compte du deuxième niveau d'indépendance s'est avérée nécessaire.

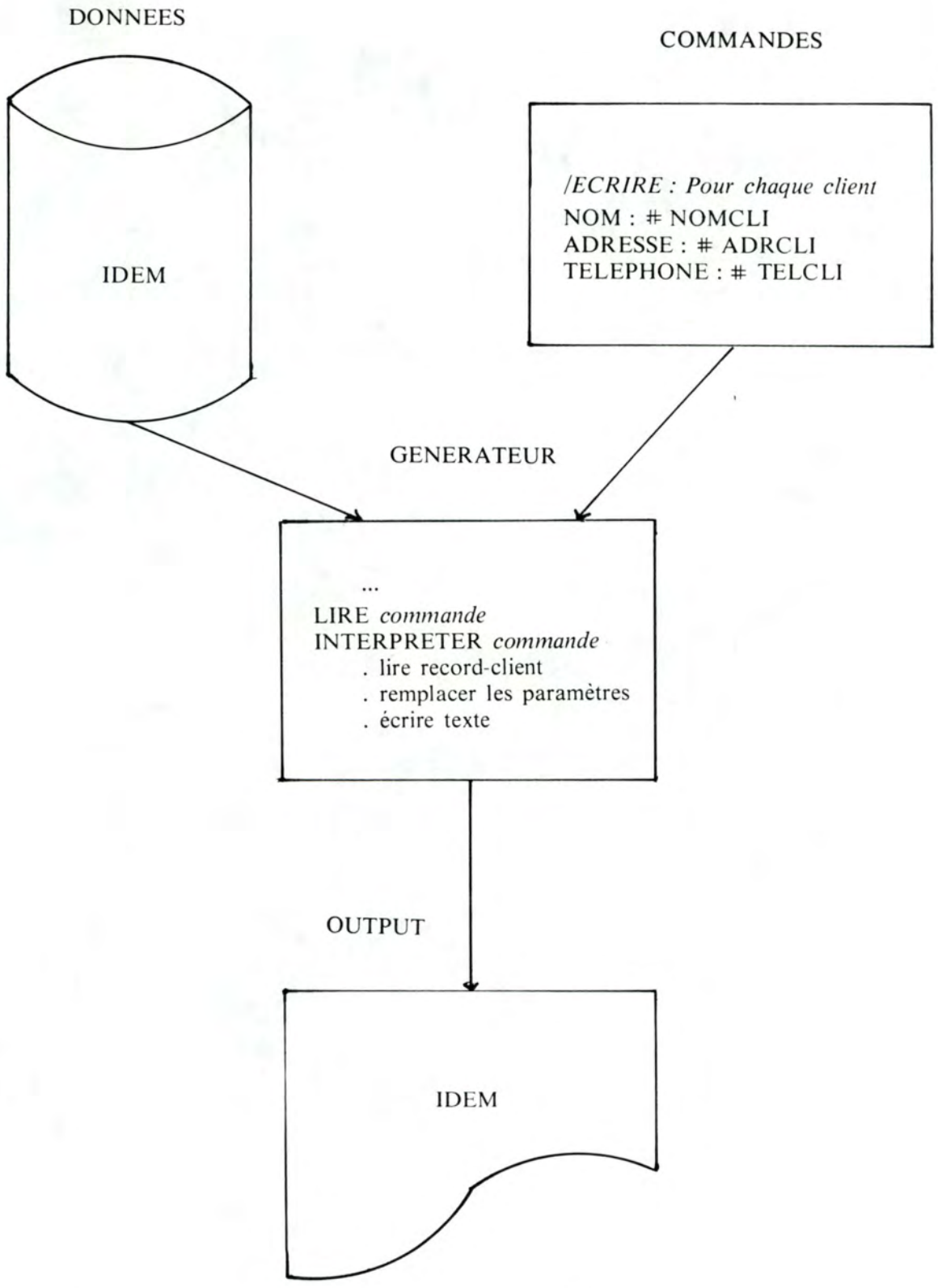
L'architecture qui fut envisagée est celle qui fut utilisée par Jean-Paul Robert [R1] dans le cadre de son mémoire :



Cette architecture diffère de la première par le fait que les commandes ne sont plus intégrées au générateur. Nous pouvons, à l'aide d'un exemple simple, comparer ces deux architectures :



Exemple d'impression de fiche client réalisé avec une architecture du niveau 1



Exemple d'impression de fiche client réalisé avec une architecture du niveau 2

Cette architecture n'a cependant pas pu être retenue tel quelle, car elle ne réalisait que partiellement l'indépendance par rapport au SGBD. A ce sujet, il convient de faire deux remarques importantes :

1. le générateur est toujours lié à la structure des données en entrée.

   En effet, le générateur est un programme. Or, d'une manière générale, tout programme contient une description de la structure des données en entrée qui correspond à la déclaration des variables (DATA DIVISION en Cobol). Ainsi, le générateur contiendra une description de la structure des données en entrée, c'est-à-dire de la description des schémas.

2. les commandes sont liées à la structure de données en entrée.

   Le langage de commandes utilisé est basé sur le même principe que le DML Codasyl. En effet, les primitives sont toujours utilisées avec un identificateur d'une donnée et non avec un identificateur d'une variable.

   Ainsi, si on trouve pour Cosasyl la primitive

   FIND record-name

   de la même manière, on trouve dans le langage de commandes des primitives telles que

   FOR-EACH record-name...

   Ce langage ne peut donc être utilisé que sur base d'un schéma de données qui est celui des données input du générateur.

Dans l'architecture proposée par Jean-Paul Robert, la structure de données utilisée était celle de Cobol. Ainsi, malgré l'avantage d'une certaine polyvalence du générateur (qui permet de générer autre chose que des interfaces), ce qui était généré restait toujours dépendant de cette structure de données. Par exemple, la notion de chemin d'accès était inconnue du générateur.

Pour remédier à cette lacune, une première tentative a été, lors de notre stage chez NCR, de généraliser la structure de données en entrée à une pseudo-structure de données du MAG. Celle-ci consistait essentiellement à introduire la notion de chemin d'accès dans la structure de données. Nous avons ainsi conçu, dans le cadre du projet DBCI [N1], un générateur réalisant le deuxième niveau d'indépendance. En effet, ce générateur fut utilisé pour générer des interfaces pour base de données gérées par IDM-9000 (COBOL avec notion de chemin d'accès) et DBS (CODASYL-78 like).

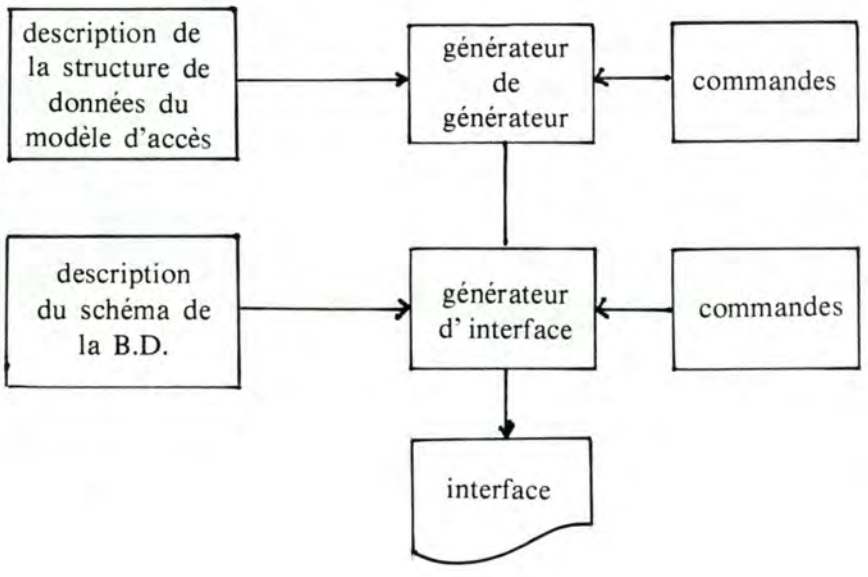
Cette structure de données, bien que relativement générale, risquait cependant de nous poser des problèmes si nous voulions réaliser un niveau d'indépendance total par rapport au SGBD. C'est pourquoi nous avons pris la structure de données du MAG qui, en fin de compte, se veut la plus générale possible. Mais

malgré son caractère général, cette structure ne permet pas de recouvrir entièrement toutes les structures de données existantes ou pouvant exister. Ainsi, nous avons été contraints de la compléter par un type de données particulier que nous avons appelé «PARTICULARITÉ».
Ce type de données n'a pas de signification (sémantique) définie à priori. Ceci est dû au fait qu'il devra contenir tout ce qui est «particulier» à une structure de données, et qui ne peut être exprimé à l'aide de celle du MAG. Par exemple : pour COBOL, le fait que des fichiers puissent être relatifs, indexés ou séquentiels, pour CODASYL, le fait qu'il existe une SET OCCURENCE SELECTION, et pour TOTAL, le fait qu'il existe des SINGLE ENTRY et des VARIABLE MASTERS. Vu que ces «particularités» peuvent se rapporter à n'importe quel élément de la structure de données du MAG, il a été indispensable d'associer ce type de données PARTICULARITÉ à chacun des éléments de la structure.

3.2.4. Troisième niveau : indépendance par rapport au modèle d'accès.

Pour ce troisième niveau, bien que non indispensable dans le cadre de ce travail, nous avons proposé une architecture qui, malgré son apparence complexe, nous paraît assez facilement réalisable. Il s'agirait en fait d'un générateur de générateur. L'avantage de cette architecture est que le générateur de générateur et les générateurs générés auraient tous la même architecture que celle exposée au § 3.2.3. La différence essentielle se situerait au niveau de la structure de données associées aux différents générateurs;

Pour le générateur de générateur, il pourrait s'agir de celle du MAG, tandis que pour les générateurs générés, elles seraient spécifiques aux modèles d'accès pour lesquels on veut les générer.



3.3 LES DONNÉES

Jusqu'à présent, nous avons présenté toutes les architectures en supposant que le générateur recevait des données, mais nous ne nous sommes pas encore intéressés à la manière dont celles-ci étaient transmises au générateur. C'est précisément le problème qui sera traité ici.
A nouveau, un certain nombre de solutions ont été envisagées avant d'aboutir à un choix définitif.

La première consistait à transmettre «manuellement» les données au générateur. Le «manuellement» pouvant se traduire par une saisie de données à l'écran :

Cette solution peut sembler simple à priori, mais s'avère en fait relativement compliquée si cette saisie doit avoir des qualités de facilité à l'utilisation.

C'est pourquoi la seconde solution, qui est celle qui fut choisie par J.P. Robert, consiste à transmettre les données à partir d'un texte contenant la définition du schéma. Ici, plusieurs cas peuvent se présenter :

1. Il existe, pour un SGBD donné, un langage de description des données généralement appelé DDL (Data Description Language).

   Dans ce cas, l'utilisateur doit définir son schéma à l'aide de ce langage, cette définition étant ensuite utilisée pour garnir le dictionnaire de données du SGBD. Il s'agit des SGBD tels que CODASYL, TOTAL, ...

   Une première solution serait que le générateur utilise directement ce dictionnaire de données. Cependant elle est peu intéressante car elle poserait de gros problèmes de portabilité et de compatibilité entre SGBD du même type.

   Une seconde solution serait de garder le principe du dictionnaire de données mais en le généralisant. Il n'existerait qu'un seul dictionnaire de données standard quel que soit le SGBD. Ce dictionnaire de données serait garni par un analyseur de la définition du schéma.
   L'inconvénient de cette solution est qu'elle nécessite la création d'un analyseur chaque fois que l'on a affaire à un nouveau SGBD et donc à un nouveau DDL.

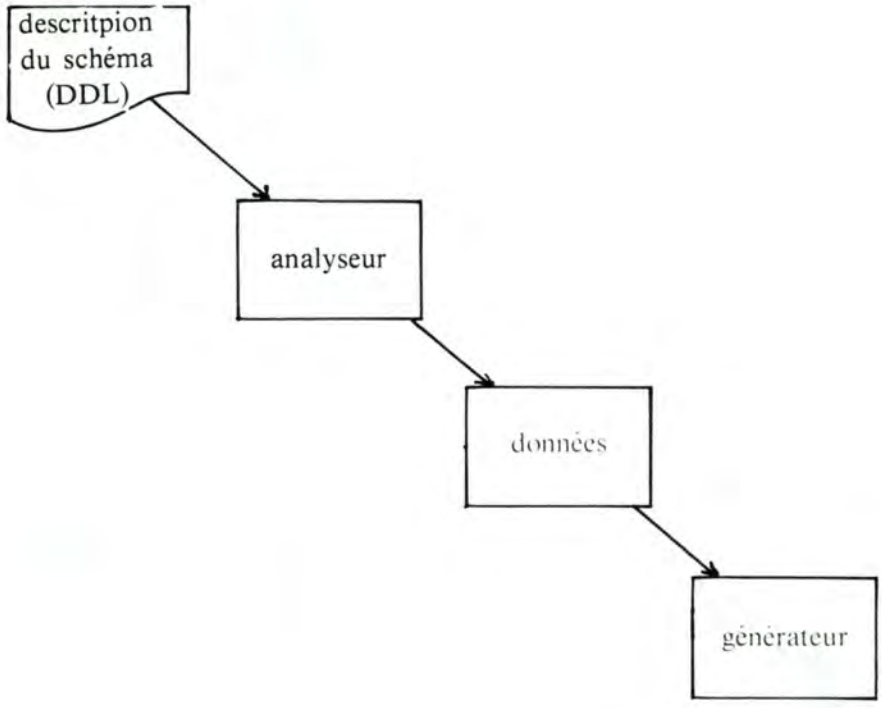
2. Il n'existe pas de langage de description des données associé au SGBD.

   C'est le cas du SGBD Cobol. Si l'on veut garder le principe du dictionnaire de données, on est forcé de concevoir à la fois le DDL et son analyseur. C'est

ainsi qu'a pratiqué J.P. Robert.

Ayant considéré ces deux cas (existence ou non d'un DDL), nous pouvons encore imaginer une dernière possibilité. En effet, cette multiplicité des DDL, et donc des analyseurs, ne s'avère guère pratique. C'est pourquoi il serait intéressant de ne disposer que d'un DDL et d'un analyseur standard pour tous les SGBD. Mais ici se repose le problème de la généralisation des structures de données des SGBD aobrdé au § 3.2.3. En effet, s'il existe un DDL pour le MAG qui se veut général, il ne permettra jamais de décrire toutes les structures de données des divers SGBD. Il faudrait donc disposer d'un analyseur du DDL-MAG et en plus, pour chaque SGBD, d'un programme permettant de communiquer au dictionnaire les «particularités» de ce SGBD. D'autre part, dans cette solution, l'utilisateur devrait pour chaque schéma le décrire avec le DDL-MAG et, s'il en existe un, avec le DDL du SGBD.

Finalement, en tenant compte de ces différents arguments, nous avons retenu la solution qui consiste à créer un analyseur par SGBD. Cette solution peut se représenter de la façon suivante :



Un élément supplémentaire nous entraîna encore à modifier quelque peu cette solution. Il s'agit de la B.D. des schémas.

Le système IDML doit disposer d'une B.D. contenant la description des schémas de toutes les B.D. qu'il supervise, tout comme les SGBD disposent d'un

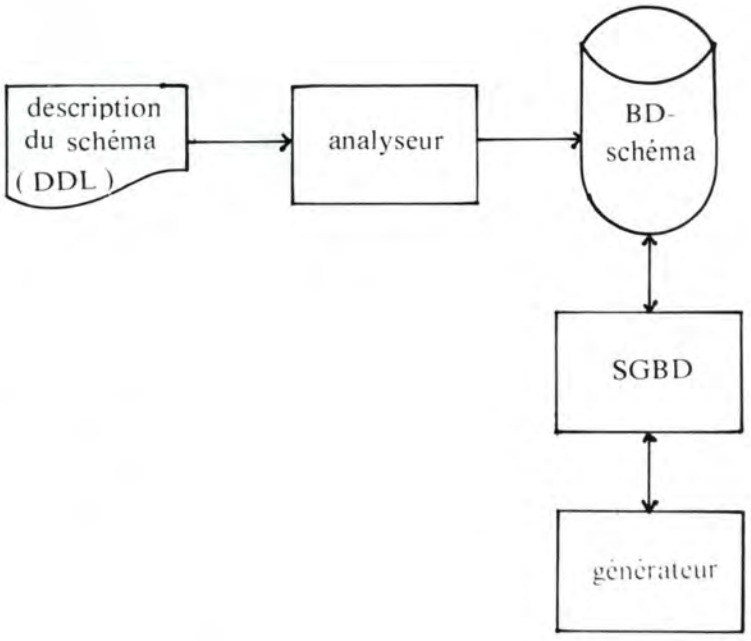
dictionnaire des données relatif aux B.D. qu'ils gèrent. Or, on sait que la description des schémas correspond aux données qui doivent être transmises au générateur.

La solution de J.P. Robert concernant la B.D. schéma était de transmettre les données obtenues à partir de l'analyse du «DDL» à la mise à jour de la B.D. schéma. Ainsi, une seule analyse du DDL était nécessaire.

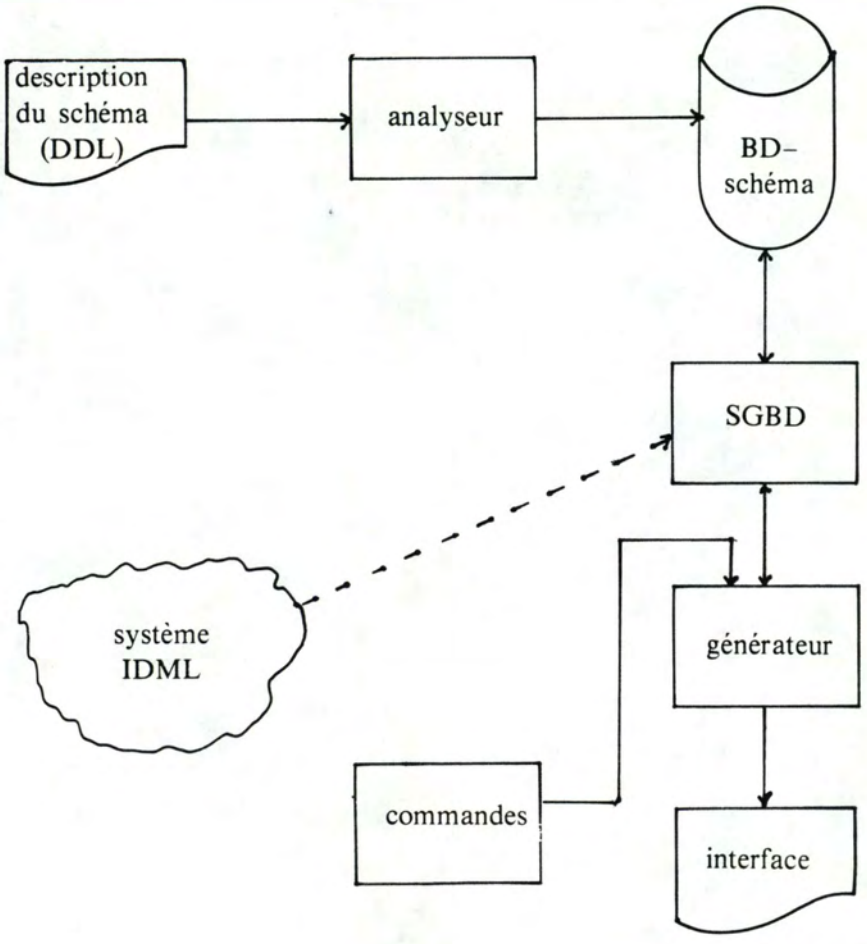
De notre côté, nous avons un peu plus approfondi le phénomène de correspondance entre les données à transmettre au générateur et celles à transmettre à la B.D. schéma, et nous avons considéré les points suivants.
Tout d'abord, la B.D. schéma n'existait pas en tant que telle; ensuite, cette B.D. schéma devait être utilisée par le système IDML, et le moyen par lequel elle serait utilisée n'était pas encore déterminé.
C'est pourquoi nous avons proposé que l'utilisation de la B.D. schéma se fasse au travers d'un SGBD offrant certaines primitives du MAG (du moins en consultation). Le chapitre IV de ce travail est consacré à la conception et à l'implantation du SGBD et de la B.D. schéma.

Pour en revenir à la transmission des données au générateur, celle-ci pourra très bien se substituer à une consultation de la B.D. schéma par le générateur, au travers du SGBD.

L'architecture de la solution finale concernant la transmission des données sera donc la suivante :



L'architecture du système de génération complet se présentera quant à lui de la façon suivante :



CHAPITRE 4. LA BASE DE DONNEES DES SOUS-SCHEMAS ET SON SGBD

Le but d'une base de données des sous-schémas est de conserver les descriptions des sous-schémas et de permettre une consultation facile de ces informations par le générateur. Additionnellement, on peut espérer que ces informations puissent servir à d'autres utilisateurs.

Pour mettre en œuvre cette méta-base de données, on a procédé en deux étapes. Dans une première étape, nous avons choisi et élaboré un SGBD qui puisse prendre en charge cette méta-base de données. Dans un deuxième temps, nous avons mis en œuvre la méta-base de données à l'aide du SGBD retenu.

4.1. LE SGBD

Après avoir exposé les considérations qui nous ont amenés à mettre en œuvre un nouveau SGBD, nous donnerons les spécifications de ce SGBD et nous expliquerons comment nous l'avons mis en œuvre.

4.1.1. Pourquoi un nouveau SGBD ?

Le programme utilisateur pour lequel la méta-base de données doit être conçue, c'est-à-dire le générateur doit pouvoir tourner sur le plus de systèmes possible. Par conséquent il est inacceptable d'avoir recours à un SGBD classique pour mettre en œuvre la méta-base de données, car on sait qu'un SGBD est souvent lié à un seul système d'exploitation.

Nous sommes donc amenés à concevoir un SGBD qui puisse d'une part satisfaire nos besoins et qui d'autre part soit facilement portable.

4.1.2. Spécifications du SGBD

Pour ne pas trop compliquer la mise en œuvre, nous sommes amenés à avoir des exigences limitées envers le SGBD à mettre en œuvre. Les exigences que nous allons formuler sont des exigences du point de vue des structures de données (4.1.1.1.) et des primitives d'accès (4.1.1.2.) et non pas d'un point de vue des performances, de la sécurité ou autres.

Lors de la formulation des exigences, nous avons essayé de rester le plus près possible du modèle d'accès généralisé afin de ne pas devoir introduire de nouveaux concepts et de nouveaux termes.

4.1.2.1. Structure de données

Avant d'exposer brièvement les structures de données à supporter par le SGBD, nous tenons à faire deux remarques.

1. Les structures de données retenues ici ont la même définition que celles qu'elles ont dans le MAG. Nous renvoyons donc le lecteur aux descriptions faites au chapitre 2.
2. Afin d'éviter une confusion entre les structures de données du SGBD à mettre en œuvre et les objets des sous-schémas à stocker dans la méta-base de données, nous appellerons les objets du SGBD «méta-objets» (ou m-objets).

L'unité logique de transfert d'information est le méta-article. Chaque méta-article appartient à un et un seul type de méta-article.

A chaque type de méta-article sont associés un ou plusieurs méta-items qui sont des types d'informations pour un ensemble de valeurs de méta-items. Les m-items sont élémentaires ou décomposables, simples et obligatoires.

La méta-clé peut être identifiante ou non. A chaque type de méta-article peuvent être associées zéro, une ou plusieurs méta-clés d'accès. Le référentiel pour la méta-clé est obligatoirement le type de méta-article d'un sous-schéma.

Un méta-chemin d'accès est un mécanisme qui associe un m-article, dit méta-origine, à zéro, un ou plusieurs m-articles, dit méta-cibles, d'une manière telle qu'il soit possible, à partir du méta-article origine, d'accéder successivement aux différentes méta-cibles.

Chaque méta-chemin appartient à un et un seul type de méta-chemin d'accès qui en définit les propriétés générales. Un type de méta-chemin n'admet pas plus d'un type de méta-article origine ou plus d'un type de méta-article cible. Un même méta-article peut être cible de plusieurs méta-chemins d'un même type.

Enfin, il faut pouvoir distinguer l'unité formée par toutes les informations relatives à un sous-schéma, car le sous-schéma sert de référentiel aux méta-clés d'accès. Comme le MAG ne connaît pas cette notion et comme d'autre part nous ne sommes pas intéressés par la notion de fichier qu'offre le MAG, nous convenons d'appeler dans la suite méta-fichier l'unité d'information formée par toutes les informations qui se rattachent à un seul sous-schéma. Ceci est possible car les primitives que nous voulons réaliser sur cette unité (cfr 4.1.2.2) sont celles que le MAG permet de faire sur les fichiers.

4.1.2.2. Primitives

Le but principal de la méta-base de données étant de permettre une consultation facile des descriptions de sous-schémas, nous avons décidé de ne retenir que des primitives d'accès.

Comme les primitives retenues sont identiques à celles décrites dans le MAG (chapitre 2), nous ne faisons que les énumérer brièvement :

- accès à la méta-base de données
- accès à un méta-fichier
- accès aux méta-articles d'un type d'un méta-fichier
- accès aux méta-articles d'un type dans un méta-chemin d'accès d'un méta-fichier
- accès aux méta-articles d'un type avec une valeur de méta-clé dans un méta-fichier.

4.1.3. Mise en œuvre du SGBD

Afin de pouvoir mettre en œuvre le SGBD, il faut trouver des représentations physiques pour toutes les structures de données requises et il faut que ces représentations soient telles qu'elles permettent une exécution facile de toutes les primitives retenues.

Pour assurer une grande portabilité de la méta-base de données, il faut que les moyens de représentation soient largement répandus.

Ces deux considérations nous ont amenés à retenir deux représentations : une première sur fichier COBOL et une deuxième en mémoire centrale. Sur les fichiers COBOL sera conservée la BD permanente. Le SGBD amène les méta-fichiers qui doivent être accédés dans des tableaux en mémoire centrale.

Ces deux représentations ont été choisies pour les avantages qu'elles représentent. Les fichiers COBOL assurent la permanence de la base de données et une relative sécurité face aux incidents. L'intérêt de la représentation en

mémoire centrale réside surtout dans des considérations de performances. Celles-ci se basent sur l'observation du comportement du générateur. En fait, le générateur d'interface consulte toujours un seul méta-fichier à la fois, mais ceci d'une façon très intensive. Consulter les fichiers COBOL à chaque demande d'accès entraînerait dans ce cas un nombre important de lectures physiques et donc une diminution des performances du système, surtout que les structures d'accès sont très complexes et ne sont pas directement prises en charge par le SGF de COBOL.

4.1.3.1. Implémentation sur fichier COBOL

La représentation d'un méta-article sur fichier comprend d'une part son identifiant et d'autre part les valeurs des méta-items rattachées à ce méta-article (fig. 4.1.1.).

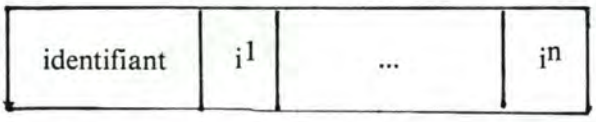


fig. 4.1.1. - Représentation d'un méta-article sur fichier.

Cet identifiant du méta-article l'identifie parmi tous les méta-articles de la méta-bd. Il est composé de :

- l'identifiant du sous-schéma (méta-fichier)
- l'identifiant du type méta-article
- l'identifiant du méta-article parmi tous les articles du même type dans le sous-schéma.

La zone des valeurs d'items (i, ... i^n) a une structure fixe pour un type de méta-article donné, car les méta-items sont obligatoires et non répétitifs.

Cette première structuration nous fournit toute l'information nécessaire pour pouvoir faire l'accès aux méta-articles d'un type d'un méta-fichier. Pour cela il suffit de parcourir le fichier en consultant l'identification de chaque méta-article enregistré. Ne sont retenus que les méta-articles dont l'identification du méta-fichier et du type correspondent à celle de la requête. Cette recherche peut être optimisée en ayant des fichiers indexés au lieu de fichiers séquentiels.

En plus, cette structure permet l'accès aux méta-articles d'un type d'un méta-fichier avec une valeur de clé. En fait, il suffit de faire le parcours décrit ci-dessus en ne retenant que les méta-articles dont l'identification correspond au méta-fichier et au type de méta-article et dont les valeurs des items qui constituent la clé correspondent à la valeur de la clé. Cette façon de procéder nous permet d'accepter un nombre non limité de clés par type de méta-article.

Cependant la structure n'est pas suffisante pour représenter un chemin d'accès et il faut donc ajouter quelque chose pour en tenir compte. Mais comment implémenter ce méta-chemin d'accès ?

Une première solution serait d'ajouter dans chaque méta-article une zone par type de méta-chemin d'accès (fig. 4.1.2.). [D4] L'ordre dans le méta-chemin est donné par l'ordre dans les pointeurs.

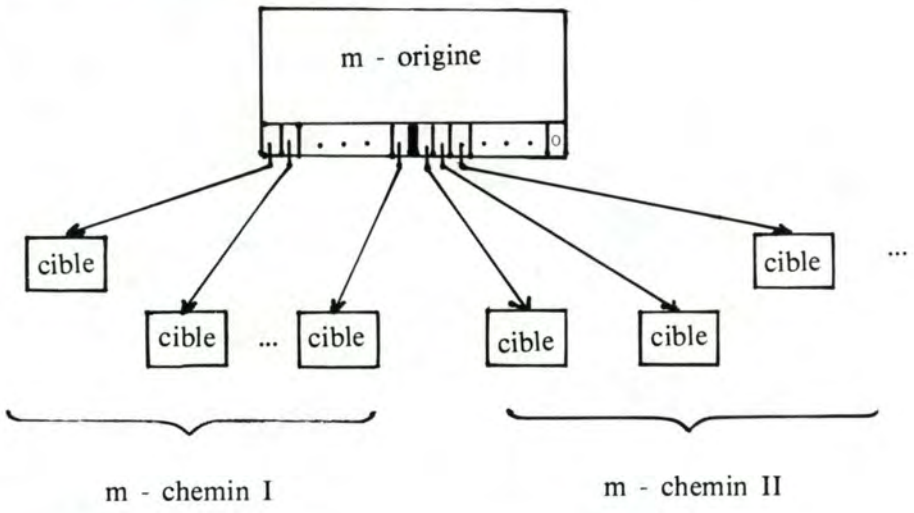


fig. 4.1.2. - Première méthode d'implémentation du chemin d'accès

Cependant cette méthode a l'inconvénient qu'il faut toujours prévoir une taille maximale pour la zone de pointeurs et ainsi on limite le nombre maximum de cibles qui peuvent participer à un chemin d'accès.

Une deuxième solution est d'avoir un pointeur de l'origine vers la première cible, puis de la première cible vers la deuxième etc. Le pointeur de la dernière cible signale la fin du chemin d'accès (fig. 4.1.3.).

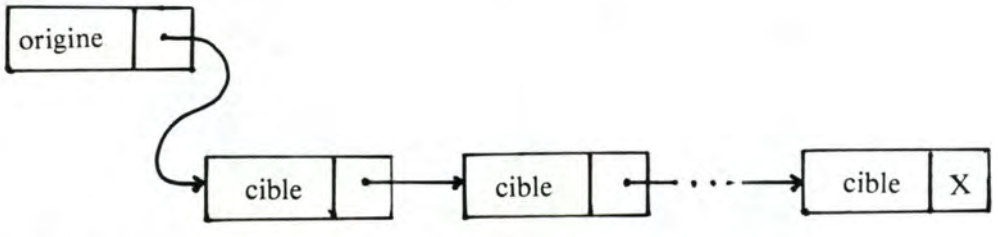


fig. 4.1.3. - Deuxième méthode d'implémentation du chemin d'accès.

Cette deuxième méthode n'est cependant pas acceptable pour nous car elle ne permet pas qu'un méta-article soit cible de plusieurs chemins d'accès du même type. En fait, il y a seulement une place de pointeur prévue par type de chemin d'accès. En réserver plusieurs pour le même type ne sert à rien puisqu'on ne pourrait de toute façon pas dire quel pointeur correspond à quelle occurrence du méta-chemin et on aboutirait donc inévitablement à une confusion de chemins.

La troisième méthode prévoit de ne plus intégrer les pointeurs dans l'enregistrement même du méta-article, mais de créer un enregistrement chaque fois qu'il y a une cible qui est insérée dans un chemin (fig. 4.1.4.).

identifiant du sous-schéma	identifiant du type de chemin d'accès	identification de l'origine	numéro d'ordre dans le chemin d'accès	identification de la cible

fig. 4.1.4. - Implémentation d'un chemin d'accès par enregistrement autonome.

Un tel enregistrement comprend :

- l'identification du méta-fichier
- l'identification du type de chemin d'accès, ce qui comprend implicitement l'identification du type d'article de l'origine et de la cible
- l'identification de l'origine parmi tous les méta-articles du même type du même méta-fichier
- l'identification de la cible parmi tous les méta-articles du même type du même méta-fichier
- un numéro d'ordre qui permet de fixer l'ordre des cibles dans un chemin d'accès.

Nous faisons la restriction que ce numéro d'ordre est attribué définitivement dès l'insertion dans le méta-chemin et donc que l'insertion se fait toujours à la fin du méta-chemin d'accès, afin qu'il n'y ait pas de perturbation de ce numéro d'ordre.

Avec ces informations, on peut accéder aux méta-articles (d'un type) dans un méta-chemin d'accès pour un méta-fichier donné. Un chemin d'accès est identifié parmi tous les autres chemins d'accès de la méta-base de données par l'identification du méta-fichier, l'identification du type de chemin d'accès et l'identification de l'origine. Cette information doit être fournie lors de la requête d'accès. Pour parcourir les cibles d'un chemin d'accès, il faut accéder à tous les enregistre-

ments de chemin d'accès qui reprennent l'identification du chemin d'accès comme décrit ci-dessus. Comme le seul ordre d'accès qu'on veut est l'ordre établi lors de la création, l'accès à ces enregistrements se fait selon l'ordre donné par le numéro d'ordre. Dès qu'on a trouvé l'enregistrement cherché, l'identification du type de la cible (donné par l'identification du type de chemin d'accès) et l'identification de la cible ainsi que l'identification du méta-fichier permettent de retrouver l'enregistrement décrivant ce méta-article.

Un dernier problème qu'il faut soulever ici est que si les méta-articles ne reprennent que l'information prévue par le modèle d'accès généralisé, on ne dispose pas de l'information pour prendre en compte les éventuelles particularités d'un SGBD ou pour faire l'émulation. Il faut qu'on prévoie dans chaque méta-article une zone assez importante afin qu'on puisse représenter les particularités qui pourraient se rattacher à ce méta-article. Il faut que cette zone soit grande et banalisée car, à priori, on ne peut pas prévoir toutes les particularités possibles. Mais comme on peut espérer que le pourcentage des méta-articles auxquels se rattachent des particularités est en général peu élevé, on est amené à avoir un important gaspillage de place. Pour éviter ce gaspillage, nous avons prévu que dans chaque enregistrement de méta-article, il n'y ait pas une zone pour les particularités mais un pointeur qui dit si oui ou non il y a une particularité rattachée à ce méta-article, et si oui, le pointeur pointe vers un enregistrement de particularité. Cet enregistrement est identifié par :

- l'identification du méta-fichier
- un numéro d'ordre.

En plus de cette identification, un article de particularités comprend un certain nombre de zones numériques et alpha-numériques dans lesquelles on peut conserver les informations nécessaires (pour le format exact, voir le «Manuel d'utilisation de la méta-base de données»).

4.1.3.2. Implémentation en mémoire centrale

Dans la partie précédente, on a vu qu'une représentation sur des fichiers COBOL peut convenir aux besoins. Cependant, il est à craindre que cette solution donne des performances médiocres dues à des accès fréquents aux fichiers.

La solution que nous adoptons ici essaie d'améliorer les performances pour un programme utilisateur spécifique : le générateur. Cependant, nous n'avons pas de connaissance suffisante des autres programmes utilisateurs pour pouvoir juger si la solution adoptée ici permet d'améliorer les performances dans le cas d'une utilisation par ces programmes-là. La solution se base sur le fait que le générateur travaille intensément sur les informations décrivant un et un seul sous-schéma. L'idée est alors d'amener toute la description de ce sous-schéma, c'est-à-

dire tout le contenu d'un méta-fichier, en mémoire centrale lorsque l'utilisateur demande l'accès à ce méta-fichier.

La description d'un méta-article se placera dans une ligne d'un tableau. Comme la longueur des méta-items varie selon le type de méta-article, on peut penser qu'il est plus intéressant d'avoir un tableau par type de méta-article. Par conséquent, il n'est plus nécessaire de reprendre l'identification du type de méta-article dans chaque ligne du tableau, puisque cette information est donnée par l'identification du tableau.

Il n'est pas non plus nécessaire de reprendre l'identification du méta-fichier si nous regroupons dans chaque tableau les méta-articles de ce méta-fichier et si nous retenons le début et la fin de cette zone relative à ce méta-fichier. Enfin, si un méta-article est identifié parmi les méta-articles d'un même type d'un méta-fichier par un nombre attribué de un à n par incréments de un, on peut ne pas le reprendre dans le tableau, à condition qu'on garnisse les lignes dans la zone du méta-fichier dans l'ordre de cet identifiant interne.

Pour accéder aux méta-articles d'un type dans un méta-fichier, il suffit de parcourir la zone correspondant au méta-fichier dans le tableau du type. L'accès par clé peut être optimisé pour une clé par type de méta-article. Si l'identifiant interne du méta-article est attribué selon l'ordre imposé par une clé d'accès, on peut effectuer pour cette clé une recherche dichotomique. Cependant, il faudra toujours faire un parcours séquentiel pour toutes les autres clés.

Comme la représentation sur fichiers des méta-chemins d'accès nous semble assez gourmande en place mémoire, nous avons adopté une autre solution. Cette solution consiste à éclater les descriptions des chemins d'accès en trois tableaux de pointeurs.

Pour éviter de trop compliquer les explications, supposons d'abord qu'on travaille avec un seul méta-fichier.

Dans le premier tableau, il y a une ligne par type de méta-chemin d'accès possible. Ces lignes sont attribuées dans un ordre déterminé, de façon à ce qu'on connaît l'entrée dans le tableau si l'on connaît le type de méta-chemin d'accès. Dans chaque ligne de ce premier tableau, il y a un pointeur vers le début d'une zone du deuxième tableau.

Dans la zone de ce deuxième tableau pointé par le pointeur du type de méta-chemin X, il y a une ligne par méta-article du type de l'origine du type X. Les liges dans cette zone sont attribuées dans l'ordre qu'ont les méta-articles dans le tableau de description de ces méta-origines. Il y a donc correspondance entre les lignes de cette zone du deuxième tableau et les lignes du tableau des méta-origines (fig. 4.1.5.).

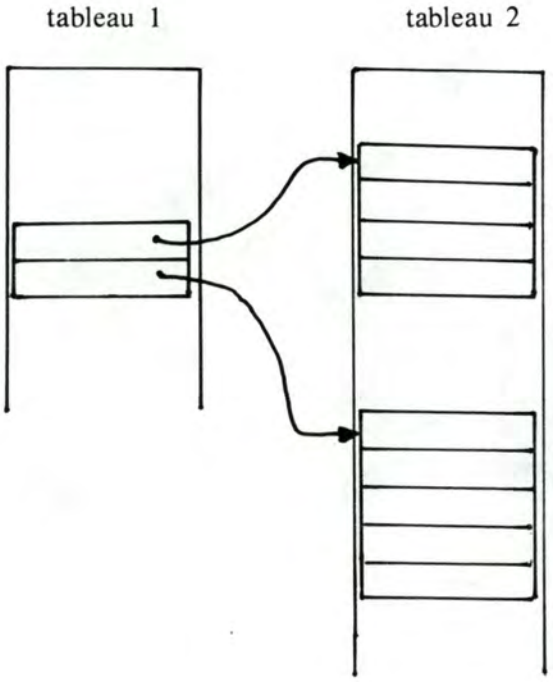


fig. 4.1.5.

Bien sûr, il faut connaître aussi la fin de chaque zone. Si l'on organise le deuxième tableau de façon à ce que la fin de la zone correspondant à un type de méta-chemin d'accès soit suivie directement par le début de la zone correspondant au type de méta-chemin suivant, il est facile de connaître la fin d'une zone. Si l'on a *n* types de méta-chemins d'accès, la fin de la dernière zone sera déterminée par un pointeur (fig. 4.1.6.).

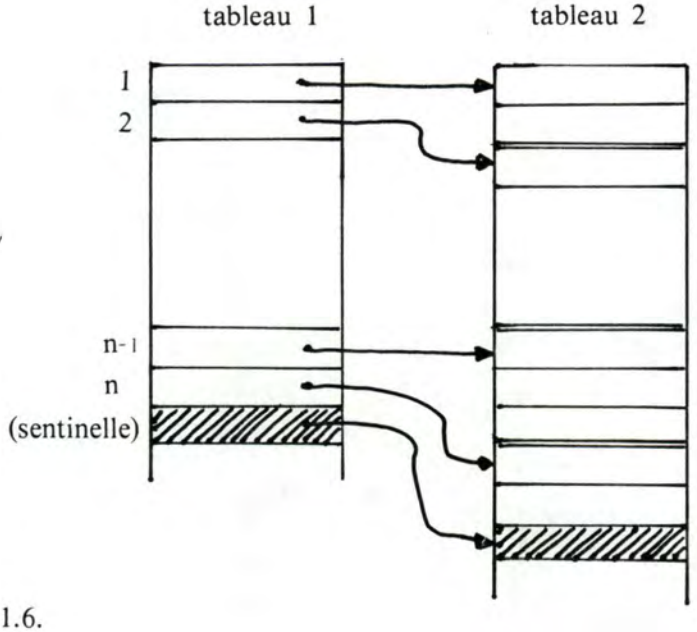


fig. 4.1.6.

Chaque ligne du deuxième tableau pointe vers une zone du troisième tableau. Une telle zone du troisième tableau contient une ligne par cible dans le méta-chemin dont l'origine est le méta-article correspondant à la ligne du deuxième tableau qui pointe vers le troisième tableau (fig. 4.1.7.).

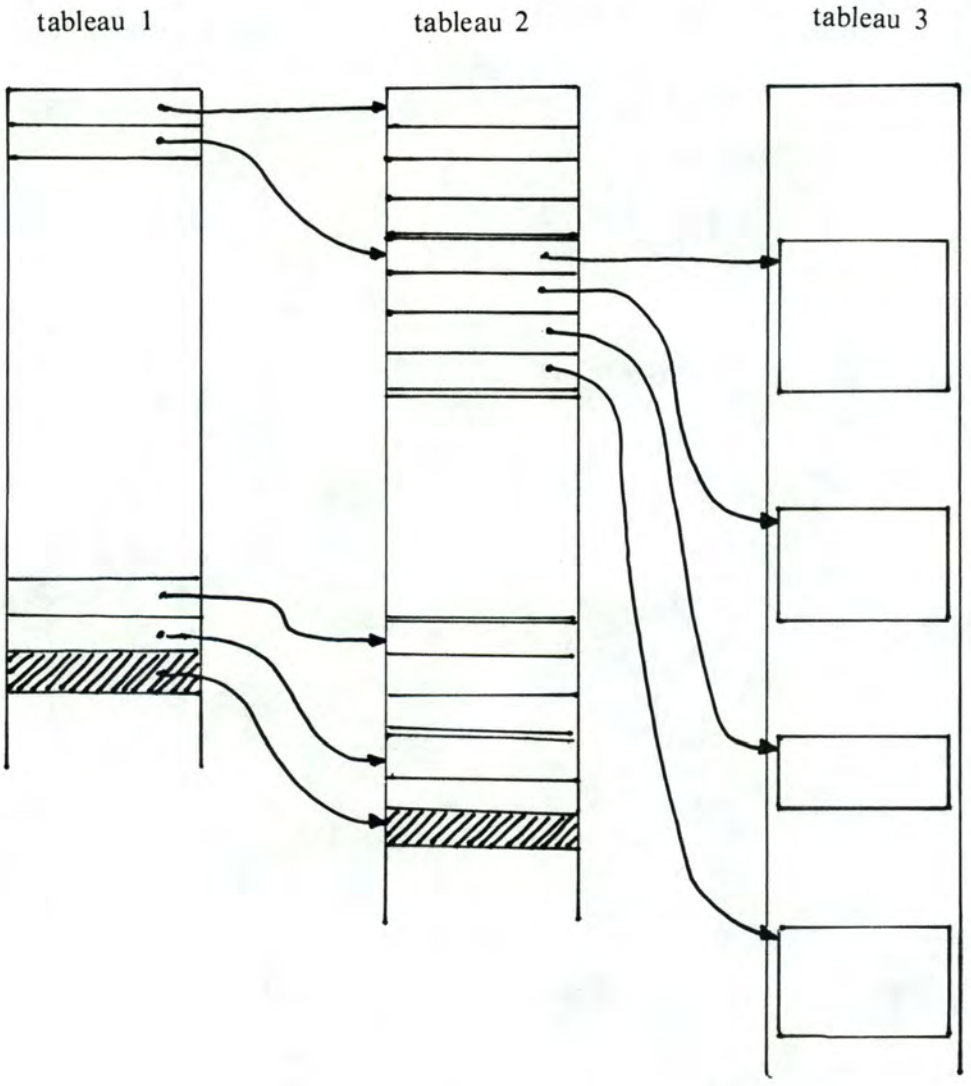


fig. 4.1.7.

Bien sûr, il faut connaître aussi la fin de chaque zone. Si l'on organise le troisième tableau de façon à ce que la fin de la zone correspondant à une méta-origine (un méta-chemin d'accès) soit suivie directement par le début de la zone correspondant à la méta-origine suivante, il est facile de connaître la fin d'une zone. La fin de la dernière zone sera déterminée par un pointeur sentinelle (fig. 4.1.8.).

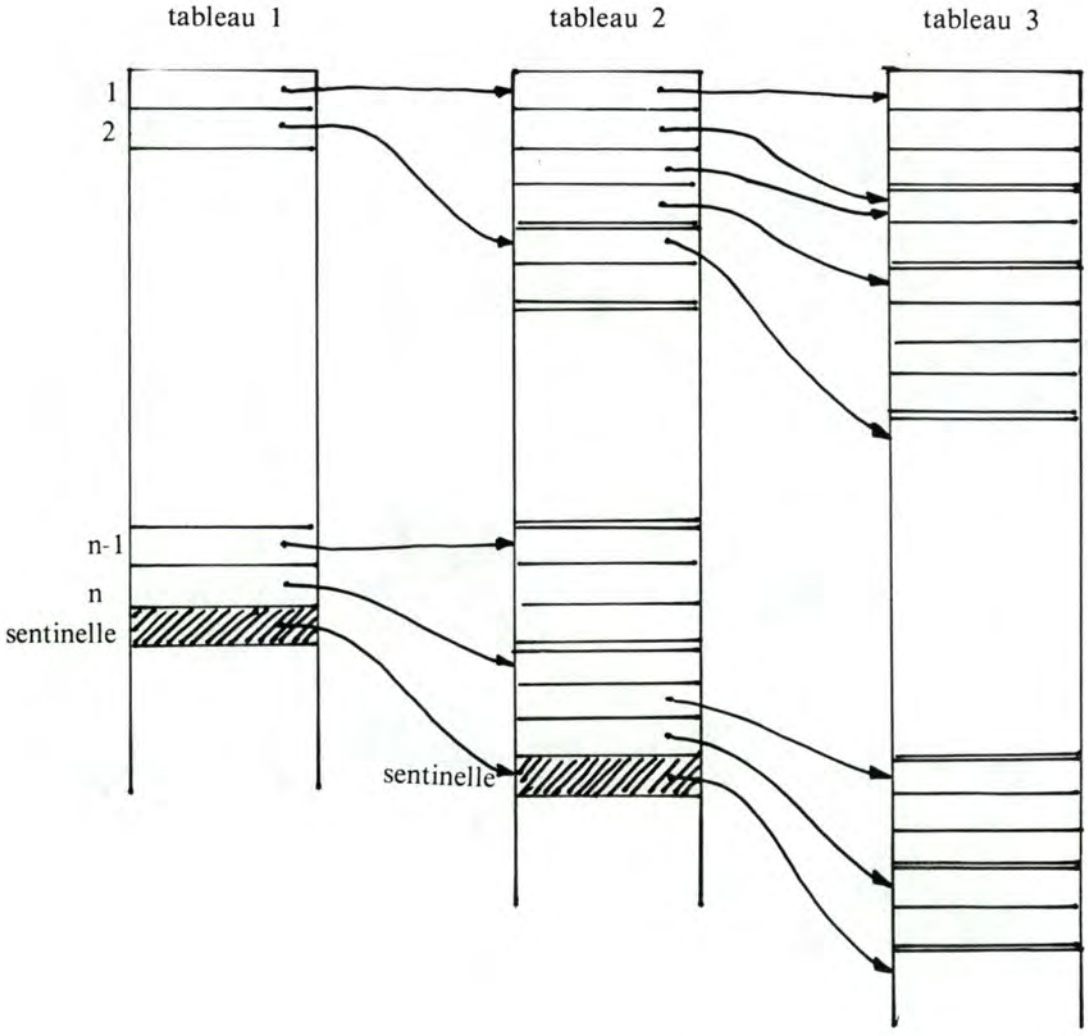


fig. 4.1.8.

Si une zone dans les tableaux deux et trois est vide, le pointeur correspondant dans les tableaux un ou deux pointe vers le début de zone prochaine. Le système reconnaît alors facilement la zone vide.

Chaque ligne du troisième tableau pointe vers une entrée dans le tableau des méta-articles du type de la cible. La ligne pointée ainsi dans ce tableau contient la description de la cible.

Pour pouvoir accepter plusieurs méta-fichiers en mémoire centrale, il suffit d'allouer une zone déterminée dans chaque tableau à chacun des méta-fichier.

Pour accéder à une cible, le système, en connaissant les paramètres d'appel, entre dans le tableau un, ce qui lui permet d'entrer dans le tableau deux, puis dans le tableau trois. Etant à la bonne ligne dans le tableau trois, il peut retrouver la cible recherchée.

4.1.4. Mise-à-jour de la méta-BD

La SGBD que nous voulons mettre en œuvre est un système «read only». Cependant, il faut que la méta-base de données puisse être mise à jour.

Si l'on veut introduire une nouvelle donnée dans la base, modifier la valeur d'un méta-item ou retirer un méta-article, il faut le faire sur les fichiers de la base permanente tout en respectant les contraintes qui sont imposées sur cette base. Les conseils pour le faire sont donnés dans le «Manuel d'utilisation de la méta-BD».

4.2. LA BASE DE DONNEES DES SCHEMAS

4.2.1. Introduction

L'objectif de cette section est d'exposer la démarche de conception de la B.D. des schémas. Cette démarche est basée sur celle qui est utilisée à l'institut d'informatique à Namur (B1, H7, H8). Cette démarche consiste en une analyse par niveau d'abstraction et comprend les étapes suivantes :

- analyse conceptuelle : à cette étape est élaboré un schéma conceptuel
- analyse logique : cette étape consiste à construire un schéma des accès logiques (clé d'accès, chemin d'accès) sémantiquement équivalent au schéma conceptuel
- analyse physique : durant cette étape, les contraintes physiques de la machine et du système sur lesquels la B.D. sera implémentée, sont prises en compte.

Les schémas élaborés au cours des différentes étapes de l'analyse sont donnés en fin de chapitre. La signification des symbolismes utilisés est reprise en annexe (cas Petit-pas : p.5).

4.2.2. Le modèle conceptuel

Le modèle qui a été choisi pour représenter le schéma conceptuel est le modèle binaire. Les différents types d'objets sont donc les types d'entités, les domaines simples et les relations. Nous donnerons tout d'abord la description de tous les types d'entités et, pour chacun de ceux-ci, la description de tous les domaines simples avec lesquels ils sont en relation. Ensuite, nous donnerons les relations entre les différents types d'entités. La démarche qui amena à ce schéma

conceptuel est développée dans un document élaboré par J.L. Hainaut (H9, H10, H11).

4.2.2.1. Les types d'entités

a) BASE-DE-DONNEES

«collection des articles d'un ensemble de fichiers et de toutes les structures qui ont été définies sur ces articles (sous-schémas)»

- les domaines simples associés :

. NOM : identificateur de la base de données qui l'identifie parmi toutes les B.D. de la B.D. des schémas
format : X(30)

. MOT-PASSE : mot de passe de la base de données
format : X(30)

. NOM-INTERFACE : identificateur de l'interface chargé de la gestion de base de données
format : X(10)

. OPER : code identifiant une opération possible sur la base de données
format : 9

. CODE : référence de la base de données
format : 99

. LGMAX : longueur du plus grand type d'articles de cette base de données; cette longueur est égale au nombre de bytes maximum associés à tous les items d'un type d'articles de la B.D.
format : 99

. NBFI : nombre de fichiers appartenant à la base de données
format : 99

. NBSI : nombre de simples appartenant à la base de données
format : 99

. NBTC : nombre de types de chemins d'accès appartenant à la base de données
format : 99

. NBTA : nombre de types d'articles appartenant à la base de données
format : 99

b) FICHIER

«collection d'ynamique d'articles»

- les domaines simples associés :

. NOM : identificateur du fichier, qui l'identifie parmi tous les fichiers de la B.D.
format : X(30)

. OPER : code identifiant une opération possible sur le fichier
format : 9

. CODE : code identifiant un fichier de la B.D.
format : 99

. NBTA : définit le nombre de types d'articles contenus dans dans un fichier
format : 99

. NBGL : définit le nombre de types de globals ayant comme référentiel le fichier décrit
format : 99

. NBSI : définit le nombre de types de simples ayant comme référentiel le fichier décrit
format : 99

c) T-ARTICLE

«définit les propriétés générales de tous les articles d'un type»

- les domaines simples associés :

. NOM : identificateur d'un type d'articles qui l'identifie à l'intérieur de la B.D.
format : X(30)

. OPER : code identifiant une opération possible sur un type d'article
format : 9

. CODE : code identifiant un type d'articles de la base de données
format : 99

. LONGUEUR : nombre de bytes occupés par tous les items associés à un type d'articles
format : 99

. NBIT : nombre d'items associés à un type d'articles
format : 99

. NBCI : nombre de cibles associées à un type d'articles
format : 99

. NBOR : nombre d'origines associées à un type d'articles
format : 99

. NBFI : nombre de fichiers associés à un type d'articles
format : 99

. NBSI : nombre de simples associés à un type d'articles
format : 99

d) ITEM

«type d'information défini par un ensemble de valeurs (domaine); décrit les propriétés des valeurs d'items qui lui son associées»

- domaines simples associés :

. NOM : identificateur d'un item, qui l'identifie dans la B.D.
format : X(30)

. CODE : code identifiant un item dans la B.D.
format : 9999
composé de :
. CODE-REL : code identifiant un item associé à un type d'article
format : 99
. CODE-TA : code identifiant le type d'articles qui contient l'item décrit.
CODE-TA ∈ T-ARTICLE (CODE);
format : 99

. OPER : code identifiant une opération possible sur un item
format : 9

. FAC/OBL : code indiquant si un item est obligatoire (O) ou facultatif (1)
format : 9

. COMPTEUR : code de l'item compteur de répétitivité de l'item décrit;
COMPTEUR∈ITEM(CODE) si l'item a une répétitivité variable
COMPTEUR = O si la répétitivité est fixe
format : 99

. REP-MAX : entier ⩾ O indiquant la répétitivité maximale d'un item
format : 999

. NO-NIVEAU : numéro de niveau d'un item (entre 2 et 49)
format : 99

. STR-INTERNE : décrit le type d'un item
0 : caractère
1 : caractère/numérique entier
2 : caractère/numérique fixe
3 : binaire entier
4 : binaire fixe
5 : réel flottant
6 : article
7 : valeur composée
format : 9

. UNITE : décrit le code interne utilisé

0 : 4 bits
1 : display-6
2 : display-7
3 : display-9
4 : mot

format : 9

. LONGUEUR : longueur d'un item en nombre d'unités du code interne utilisé

format : 999

. POS-DECIM : donne le positionnement du point décimal

format : 99

. NBIT : Nombre d'items associés à un item (descendants)

format : 99

. NBCO : nombre de composants associés à un item

format : 99

e) ORIGINE

«représente la participation comme origine d'un type d'articles à un type de chemins»

- domaines simples associés :

. MODE-INS : définit le mode d'insertion d'une origine dans un type de chemins :

0 : automatique
1 : manuel

format : 9

. MODE-RETR : définit le mode de retrait d'une origine d'un type de chemins :

0 : optionnel
1 : obligatoire
2 : fixe

format : 9

. OPER : code identifiant une opération possible sur une origine

format : 9

f) CIBLE

«représente la participation comme cible d'un type d'articles à un type de chemins»

- domaines simples associés :

. MODE-INS : définit le mode d'insertion d'une cible dans un type de chemins :

0 : automatique
1 : manuel

format : 9

. MODE-RETR : définit le mode de retrait d'une cible d'un type de chemins :

0 : optionnel
1 : obligatoire
2 : fixe

format : 9

. OPER : code identifiant une opération possible sur une cible
format : 9

g) T-CHEMIN

«définit les propriétés communes de tous les chemins d'un même type»

- domaines simples associés :

. NOM : identificateur d'un type de chemins d'accès, qui l'identifie dans la B.D.
format : X(30)

. CODE : code identifiant un type de chemins d'accès de la B.D.
format 99

. CONNECTIVITE : connectivité d'un type de chemins d'accès :

0 : 1-N
1 : N-1
2 : M-N

format : 9

. INVERSE : code identifiant le type de chemins inverse du type de chemins décrit :

INVERSE ∈ T-CHEMIN(CODE) ou
INVERSE = 0 ce qui signifie qu'il n'existe pas de chemin inverse

format : 99

. NBOR : nombre d'origines associées à un type de chemins
format : 99

. NBCI : nombre de cibles associées à un type de chemins
format : 99

. NBGL : nombre de globaux ayant pour référentiel un type de chemins
format : 99

. NBSI : nombre de simples ayant pour référentiel un type de chemins
format : 99

. NBCO : nombre de composants associés au type de chemins
format : 99

h) GLOBAL

«décrit un mécanisme «global» dont la classe peut être :
- identifiant et/ou
- ordre et/ou
- clé d'accès

Le terme «global signifie que les notions d'identifiant, ordre ou clé d'accès sont définies sur tous les articles du référentiel qui leur est associé».

- domaines simples associés :

. CLASSE : définit le type de global décrit;
composé de :
. CLASSE-KEY : = 1 si le global correspond à une clé d'accès
0 sinon
format : 9
. CLASSE-ORDRE : = 1 si le global correspond à un ordre
0 sinon
format 9
. CLASSE-IDENT : = 1 si le global correspond à un identifiant
0 sinon
format : 9

. TYPE-REF : type de référentiel associé au global :
0 : base de données
1 : fichier
2 : type de chemins
format : 9

. ORDRE : décrit l'ordre des éléments :
0 : sans objet
1 : non significatif
2 : chronologique (FIFO)
3 : anti-chronologique (LIFO)
4 : programmé (PRIOR)
5 : programmé (NEXT)
6 : trié (SORTED)
7 : par type d'articles
format : 9

. CODE : code identifiant un GLOBAL :
format : 99

. DOUBLE : détermine le traitement des doubles :
0 : sans objet
1 : non significatif
2 : chronologique (LAST)
3 : anti-chronologique (FIRST)
si CLASSE = 0 (identifiant) alors DOUBLE = 0
format : 9

i) SIMPLE

«décrit un mécanisme «simple» dont la classe peut être :
- identifiant et/ou
- ordre et/ou
- clé d'accès

Le terme «simple» signifie que les notionsd'identifiant, ordre ou clé d'accès sont définies sur les articles du référentiel qui leur est associé, qui appartiennent à un type particulier».

- domaines simples associés :

. CLASSE : définit le type de simple décrit;
composé de :

. CLASSE-KEY : = 1 si le simple correspond à une clé d'accès
0 sinon
format : 9

. CLASSE-ORDRE : = 1 si le simple correspond à un ordre
0 sinon
format : 9

. CLASSE-IDENT : = 1 si le simple correspond à un identifiant
0 sinon
format : 9

. TYPE-REF : type de référentiel associé au simple :
0 : base de données
1 : fichier
2 : type de chemins
3 : global
format : 9

. ORDRE : décrit l'ordre des éléments :
0 : sans objet
1 : non significatif

2 : chronologique (FIFO)
3 : anti-chronologique (LIFO)
4 : programmé (PRIOR)
5 : programmé (NEXT)
6 : trié (SORTED)
format : 9

. CODE : code identifiant un SIMPLE
format : 9

. DOUBLE : détermine le traitement des doubles :
0 : sans objet
1 : non significatif
2 : chronologique (LAST)
3 : anti-chronologique (FIRST)
si CLASSE-IDENT = 1 (identifiant) alors DOUBLE = 0
format : 9

j) COMPOSANT

«représente la participation d'unitem ou d'un type de chemin à un simple en tant que composant de celui-ci»

- domaines simples associés :

. CODE : code identifiant uncomposant
format : 99

. TYPE-IT/TC : indique si le composant est associé à unitemou à un type de chemins :
0 : item
1 : type de chemins
Si le simple associé au composant correspond à un identifiant (CLASSE-IDENT = 1)
alors TYPE-IT/TC peut valoir 0 Ou 1.
Si le simple associé au composant correspond à une clé d'accès ou à un ordre (CLASSE-KEY = 1 ou CLASSE-ORDRE = 1)
alors TYPE-IT/TC doit valoir 0.
format : 9

. SENS : détermine le sens d'un ordre sur un item :
0 : sans objet
1 : croissant
2 : décroissant

Si le simple associé au composant correspond à un ordre trié (CLASSE-ORDRE = 1 et ORDRE = 6)

alors

SENS doit valoir 1 ou 2

Dans tous les autres cas, SENS = 0

format : 9

4.2.2.2. Les relations entre types d'entités

- FISI (FICHIER, SIMPLE)	(f, s) : le simple (S) a pour référentiel les articles du fichier (f)
- FIGL (FICHIER, GLOBAL)	(f, g) : le global (g) a pour référentiel les articles au fichier (f)
- TAFI (T-ARTICLE, FICHIER)	(t, f) : le type d'article (t) est affecté au fichier (f)
- TAIT (T-ARTICLE, ITEM)	(t, i) : à l'item (i) est associé un type d'articles (t)
- TAOR (T-ARTICLE, ORIGINE)	(t, o) : le type d'articles (t) participe à un chemin d'accès en tant qu'origine (o)
- TACI (T-ARTICLE, CIBLE)	(t, c) : le type d'articles (t) participe à un chemin d'accès en tant que cible (c)
- TASI (T-ARTICLE, SIMPLE)	(t, s) : les articles du référentiel du simple (s) sont du type (t)
- ITIT (ITEM, ITEM)	(i, j) : l'item (j) est composant de l'item (i)
- ITCO (ITEM, COMPOSANT)	(i, c) : l'item (i) participe à un simple comme composant (c)
- TCOR (T-CHEMIN, ORIGINE)	(t, o) : le type de chemins (t) a pour origine (o)
- TCCI (T-CHEMIN, CIBLE)	(t, c) : le type de chemins (t) a pour cible (c)

- TCCO (T-CHEMIN, COMPOSANT)	(t,c) : le type de chemins (t) participe à un simple en tant que composant(c)
- TCSI (T-CHEMIN, SIMPLE)	(t, s) : le simple (s) a pour référentiel les articles cibles du type de chemins (t)
- TCGL (T-CHEMIN, GLOBAL)	(t,g) : le global (g) a pour référentiel les articles cibles du type de chemins (t)
- SICO (SIMPLE, COMPOSANT)	(s, c) : le simple (s) est composé du composant (c)
- GLSI (GLOBAL, SIMPLE)	(g,s) : le simple (s) a pour référentiel celui du global (g)
- BDSI-1 (BASE-DE-DONNEES, SIMPLE)	(b, s) : le simple (s) a pour référentiel tous les articles de la base de données (b).

Il faudrait encore ajouterà cette liste toutes les relations entre BASE-DE-DONNEES et chacun des éléments de la base de données.

4.2.3. Le modèle des accès logiques

A partir du schéma conceptuel, nous pouvons construire un schéma des accès logiques, sémantiquement équivalent à celui-ci, et qui met en évidence les accès aux données. Les données du modèle des accès logiques (H13) sont décrites en termes de fichiers, articles, valeurs d'item, identifiants d'un type d'articles, clés d'accès et chemins d'accès. Les objets de base à accéder sont les articles et les valeurs d'item. Les mécanismes d'accès sont d'un article à des valeurs d'item, de valeurs d'item à un article (clé d'accès) et d'un article à des articles (chemin d'accès). Un chemin d'accès est orienté.

Le passage du modèle conceptuel binaire au modèle des accès logiques peut être fait aisément. Aux domaines simples on fera correspondre les items, aux domaines-entités des types d'articles et aux relations, des mécanismes d'accès.

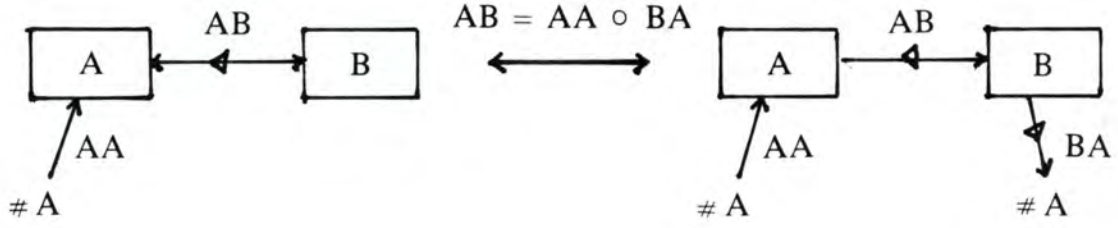
En principe, il convient de distinguer les accès possibles et les accès nécessaires, ces derniers étant définis par les modules fonctionnels qui travailleront sur la base de données. Dans notre cas, nous ne connaissons que certains utilisateurs potentiels de cette base de données, à savoir le système IDML et le générateur. Ainsi, afin de pouvoir faire face à toute demande d'accès émanant d'autres utilisateurs éventuels, nous avons voulu rester aussi général que possible

en fournissant tous les accès possibles. De cette manière, il n'y aura pas de différence entre le schéma des accès possibles et celui des accès nécessaires.

Remarque : la relation de descendance ITIT (ITEM, ITEM) a été remplacée par un chemin d'accès N-1 (item-fils - item-père), le chemin inverse n'étant pas réalisé. Cette relation de descendance se traduira partiellement par une contrainte qui impose un ordre sur les cibles du chemin d'accès de T-ARTICLE vers ITEM, cet ordre étant l'ordre dynastique de la structure de décomposition des items.

4.2.4. Le modèle du SGBD

Du fait que le SGBD a été conçu en fonction du schéma des accès logiques, le modèle du SGBD ne diffèrera que de peu de celui-ci. En effet, la différence essentielle est que, pour un souci de performance, tous les chemins N-1 ont été transformés de la manière suivante :



$$BA\,[\,\#\,A\,] \subseteq AA\,[\,\#\,A\,]$$

De ce fait, un certain nombre d'items supplémentaires ont été associés aux types d'articles origines de chemin N-1 :

- ITEM

  . CODE-IT : code de l'item père
  S'il existe, alors CODE-IT ∈ ITEM(CODE)
  Sinon (item du niveau le plus bas (02) CODE-IT = 0
  format : 99

- ORIGINE

  . CODE-TC : code du type de chemins dont l'origine est décrite
  CODE-TA ∈ T-ARTICLE(CODE)
  format : 99

- CIBLE

. CODE-TC : code du type de chemins dont la cible est décrite
CODE-TC ∈ T-CHEMIN(CODE)
format : 99

. CODE-TA : code du type d'articles correspondant à la cible
CODE-TA ∈ T-ARTICLE(CODE)
format : 99

- GLOBAL

. REFER : code identifiant l'élément dont le type est donné par TYPE-REF
Si TYPE-REF = 0
alors REFER ∈ BASE-DE-DONNEES(CODE)
Si TYPE-REF = 1 alors REFER ∈ FICHIER(CODE)
ou REFER = 0, ce qui signifie que la référence est étendue à tous les fichiers de la B.D.
Si TYPE-REF = 2 alors REFER ∈ T-CHEMIN(CODE)
format : 99

- SIMPLE

. REFER : code identifiant l'élément dont le type est donné par TYPE-REF
Si TYPE-REF = 0
alors REFER ∈ BASE-DE-DONNEES(CODE)
Si TYPE-REF = 1 alors REFER ∈ FICHIER(CODE)
ou REFER = 0, ce qui signifie que la référence est étendue à tous les fichiers de la B.D.
Si TYPE-REF = 2 alors REFER ∈ T-CHEMIN(CODE)
Si TYPE-REF = 3 alors REFER ∈ GLOBAL(CODE)
format : 99

. CODE-TA : code identifiant le type d'article associé à un simple
CODE-TA ∈ T-ARTICLE(CODE)
format : 99

- COMPOSANT

. CODE-SI : code identifiant le simple associé à un composant

CODE-SI $\in$ SIMPLE(CODE)

format : 99

. CODE-IT-TC : code identifiant l'item ou le type de chemins associé à un composant

Si TYPE-IT/TC = 0 alors CODE-IT/TC $\in$ ITEM(CODE)

Si TYPE-IT/TC = 1 alors CODE-IT/TC $\in$ T-CHEMIN(CODE)

format : 9

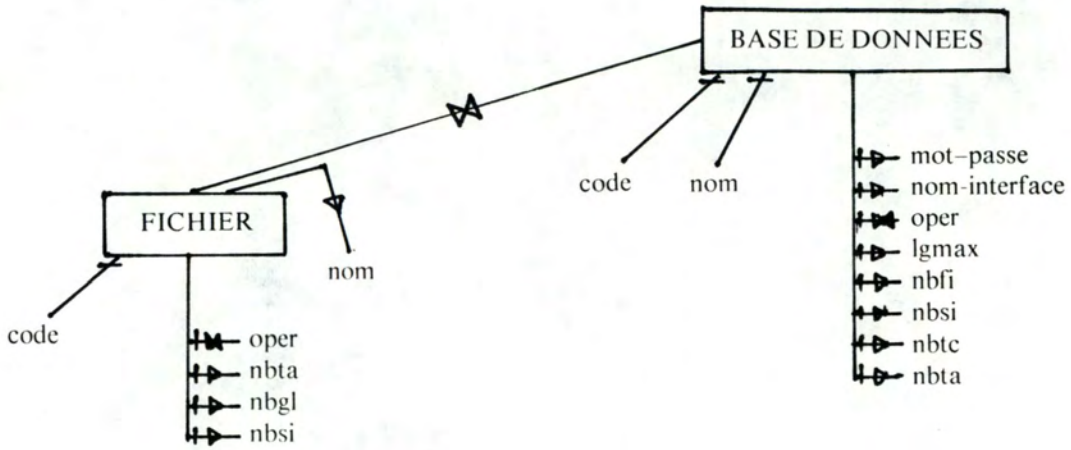


Schéma conceptuel de la BD-schéma

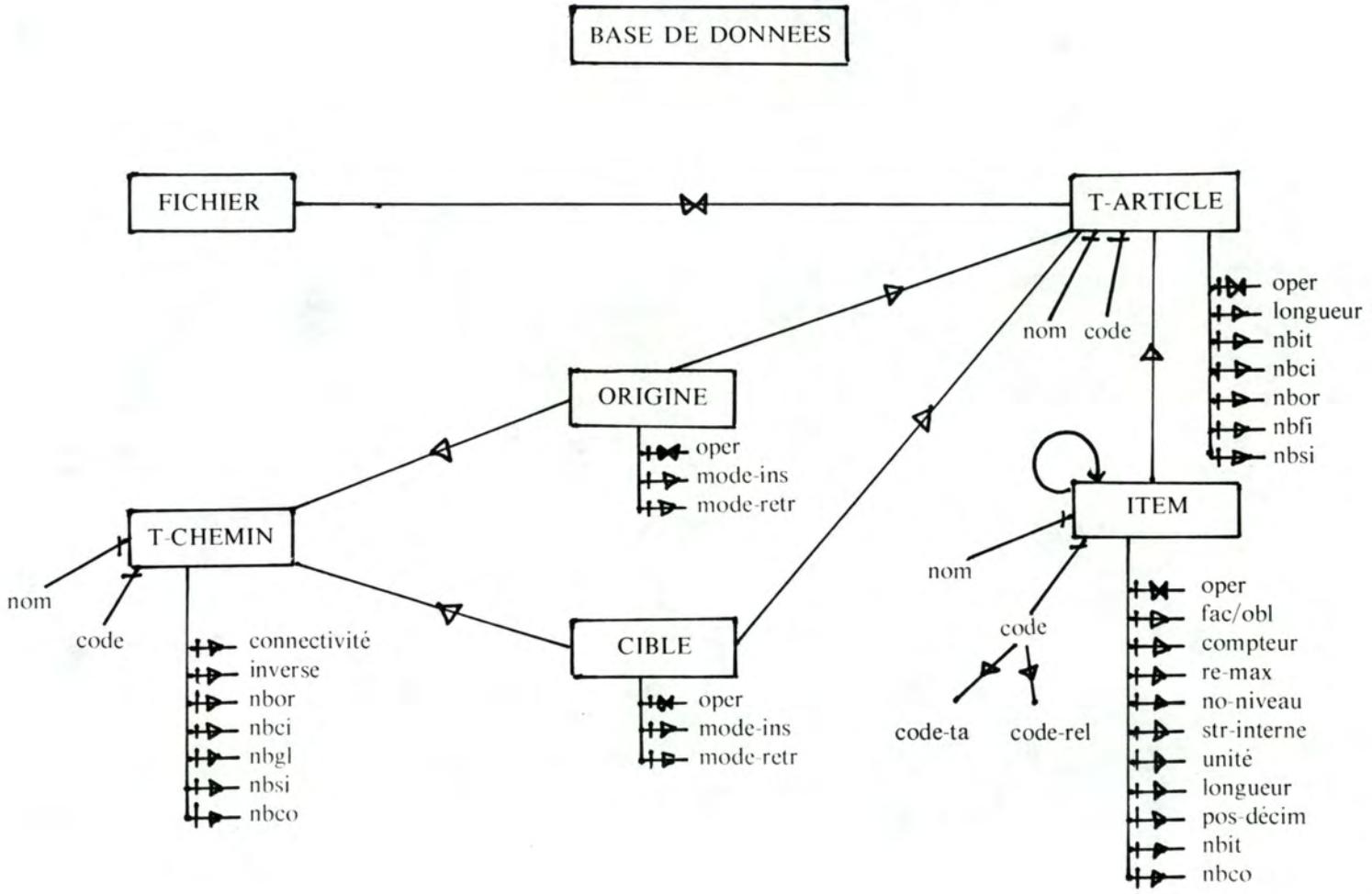


Schéma conceptuel de la BD-schéma (suite).

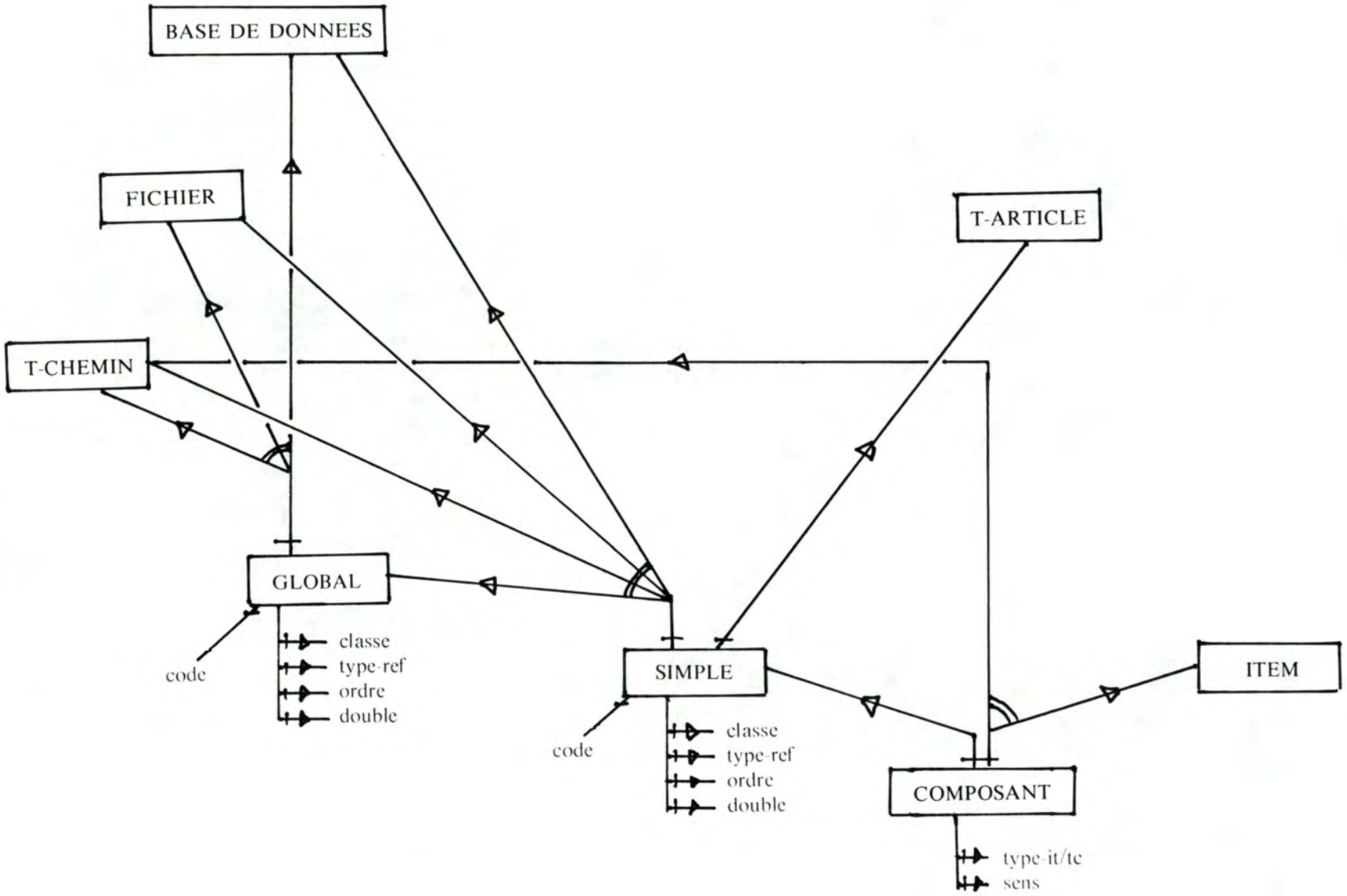


Schéma conceptuel de la BD-schéma (suite).

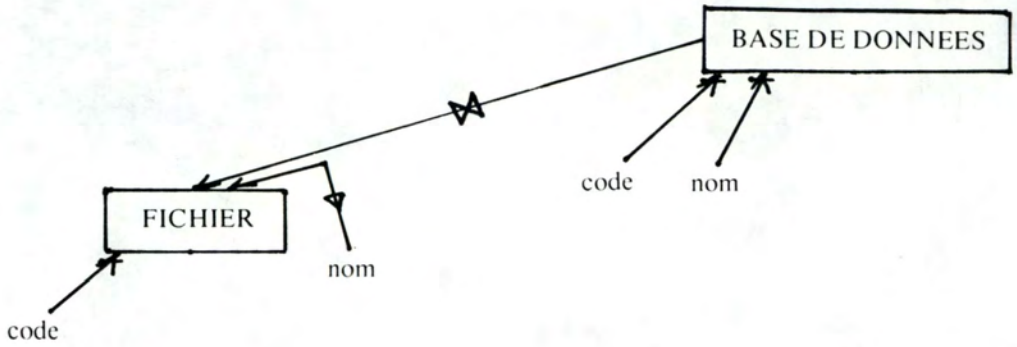


Schéma des accès logiques de la BD-schéma (schéma du SGBD).

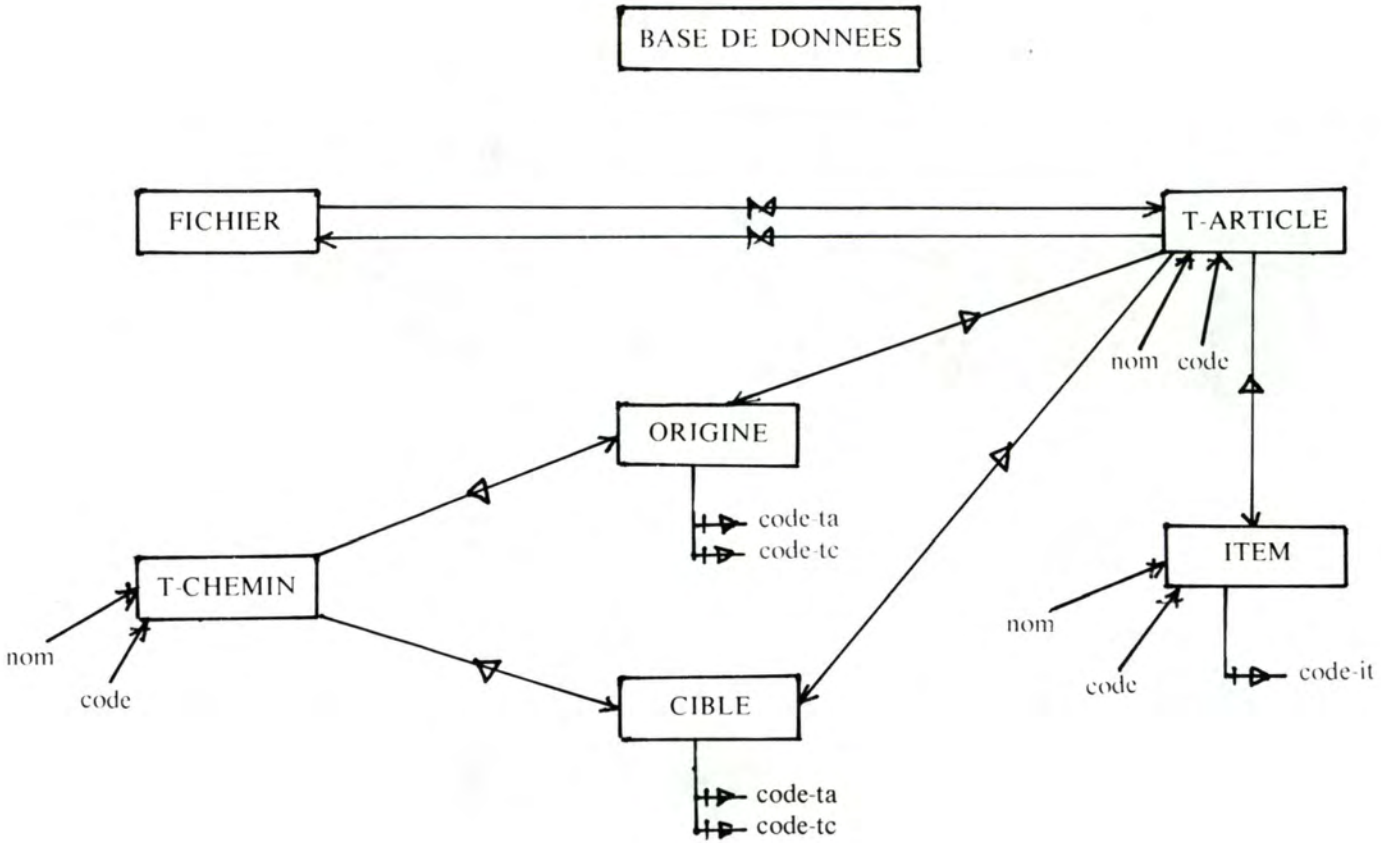


Schéma des accès logiques de la BD-schéma (schéma du SGBD) (suite).

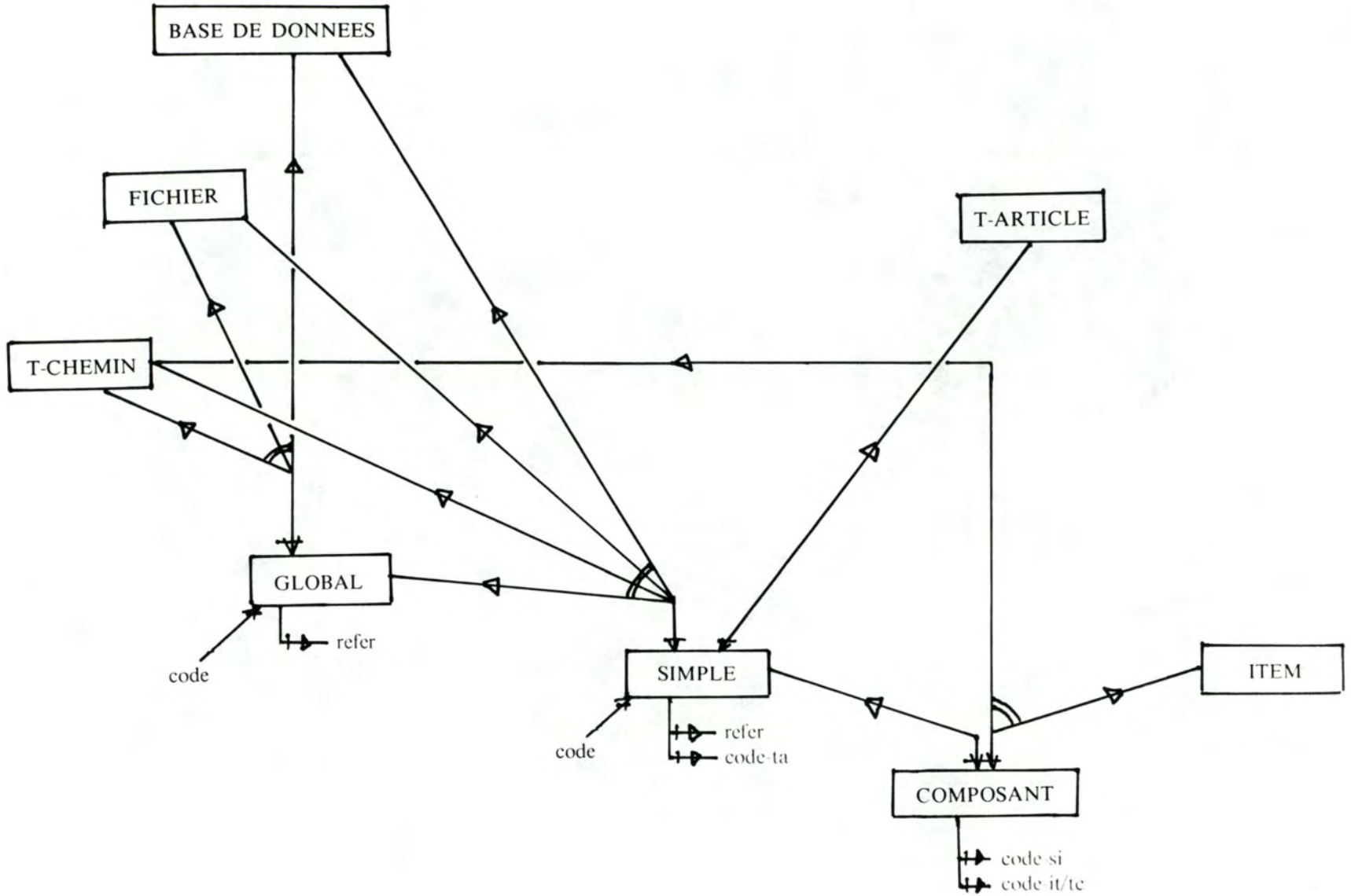


Schéma des accès logiques de la BD-schéma (schéma du SGBD) (suite).

CHAPITRE 5. : LA GENERATION

Pour pouvoir générer une interface pour un sous-schéma dans un SGBD donné, le générateur doit avoir accès d'une part à des informations concernant le SGBD et d'autre part, à des informations concernant le sous-schéma. Il trouvera ces informations dans les commandes de génération et dans la méta-base de données (fig. 5.0.1.).

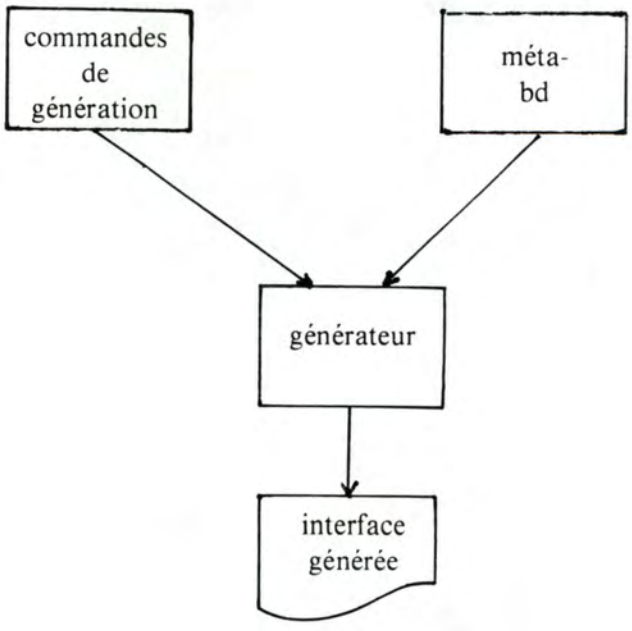


fig. 5.0.1. – Architecture du système de génération

Dans le chapitre précédent, la méta-base de données et son SGBD ont été décrits. Dans la première partie de ce chapitre-ci, nous décrirons les commandes de générations.

Dans la deuxième partie du chapitre sera décrit sommairement comment le générateur produit une interface à partir des informations qu'il a reçues à l'entrée, c'est-à-dire d'une part les commandes de génération et d'autre part la description du sous-schémas contenu dans la méta-base de données.

Enfin, dans une troisième partie, nous essayerons d'adopter un point de vue critique vis-à-vis des options prises et de faire des suggestions d'amélioration par rapport à la solution actuelle.

5.1. LE LANGAGE DE COMMANDE DE GENERATION

5.1.1. Introduction

Un langage de commande de génération doit satisfaire à des objectifs souvent contradictoires. D'abord il faut que ce langage ait la puissance suffisante pour qu'on soit à même de générer tout ce dont on a besoin. Ensuite, il faut que le langage soit facile à apprendre et à utiliser. Et enfin, il faut que le langage ne soit pas trop compliqué à exploiter par le générateur. Malheureusement, un langage puissant et agréable pour l'utilisateur risque de ne pas être facilement exploitable (et il s'agit donc de trouver un compromis entre les différents objectifs).

Le langage de commande de génération que nous avons mis en œuvre (et qui est une version améliorée de GECOL [N1]), est né d'un compromis entre ces trois objectifs, tout en donnant une certaine priorité à l'objectif de puissance.

Lors du développement de ce langage, il est apparu qu'on peut distinguer deux types de commandes.

Le générateur est supposé produire un texte dans un langage de programmation (en l'occurrence COBOL). Pour le faire, il doit disposer d'une image de ce texte. En premier lieu, on trouvera donc des morceaux de programmes dans un texte de commandes de génération (macros). Cependant, certaines parties du texte (notamment certains identifiants), qu'on voudra retrouver dans une interface générée, ne peuvent pas être connues à la rédaction des macros, car elles sont propres à un sous-schéma particulier. Dans le texte des commandes, on va trouver une identification du type de la partie de texte à remplacer, qu'on appelera paramètre par la suite, et le générateur se chargera de la remplacer par la valeur correspondante dans le sous-schéma courant. Un morceau de texte, composé de parties de texte fixes et de parties de texte à remplacer ou au moins d'un des deux, sera appelé une commande de génération (macros).

Pour garantir que le texte à la sortie soit bien assemblé et que les remplacements de texte soient faits, il faut des commandes à l'aide desquelles on peut diriger la génération. Ces commandes sont appelées directives (de génération).

En plus de ces deux types de commandes, on peut se donner la possibilité d'insérer du commentaire dans un texte de commande de génération.

Dans la suite, nous exposerons plus en détails les possibilités du langage de commande de génération, sans aller cependant jusqu'à exposer le format du langage. Le lecteur trouvera celui-ci dans le manuel de l'utilisateur du générateur (Annexe).

5.1.2. Les directives de génération

a) Boucle

Il est souvent nécessaire de répéter dans un texte généré une partie de texte pour chaque occurrence d'un méta-type d'article, en changeant seulement à chaque fois les parties variables du texte à l'entrée. Il faut donc se donner la possibilité de boucler plusieurs fois sur les mêmes commandes (fig. 5.1.1.).

for-each type de méta-article **do** < texte > **od**

fig. 5.1.1. – boucle de génération (format 1)

Afin que le remplacement du texte puisse se faire, la directive de boucle assure, en plus de la répétition de texte à la sortie, que l'accès se fait au bon méta-article. L'ordre d'accès est celui offert par la méta-base de données.

Cependant, il peut parfois s'avérer insuffisant d'avoir la possibilité de boucler sur toutes les occurrences d'un même méta-type. On voudrait pouvoir boucler seulement sur toutes les cibles d'un méta-chemin d'accès défini par la méta-base de données. Ainsi par exemple, on veut générer un morceau de texte pour tous les types de cibles d'un type de chemin d'accès, dans le méta-chemin d'accès «type de chemin d'accès vers type de cible».

for-each type de méta-article **within** type de méta-chemin
do < texte > **od**

fig. 5.1.2. – boucle de génération (format 2)

Accéder à toutes les cibles d'un chemin d'accès veut tout d'abord dire qu'on connaît le chemin d'accès. Un méta-chemin d'accès est identifié par le type du méta-chemin d'accès et par l'occurrence de l'origine. Il faut donc avoir préalablement accédé à l'origine et il y a par conséquent une nécessité de pouvoir imbriquer des boucles de génération.

Enfin, on voudrait pouvoir générer un texte pour tous les articles d'un méta-type qui ont une même valeur de méta-clé. Cette valeur peut être connue au moment de la rédaction des macros, mais elle peut aussi être une valeur accédée dans une boucle extérieure et être donc seulement connue lors de la génération (fig. 5.1.3.).

for-each type de méta-article **using** nom de méta-clé

equal { littéral / identifiant } **do** < texte > **od**

fig. 5.1.3. – boucle de génération (format 3)

Il est évident qu'on peut seulement prendre des clés qui sont définies dans le schéma d'accès de la méta-base de données.

Il pourrait être intéressant de disposer d'autres opérateurs que le «égal à». Cependant, d'une part l'accès par clé offert par l'interface de consultation de la méta-base de données ne prévoit que le «égal à» comme opérateur, et donc le générateur devrait simuler les autres, et d'autre part, dans les problèmes de génération que nous avons rencontrés, nous n'avons pas ressenti le besoin d'autres opérateurs. Si toutefois le besoin devait apparaître, on peut résoudre le problème en utilisant une boucle simple, suivie d'une sélection (voir plus loin) sur les méta-items composant la clé d'accès, puisque la sélection offre d'autres opérateurs, comme on le verra.

Bien que ce mécanisme d'accès par boucle est assez puissant, il existe des cas où il n'est pas suffisant. Ainsi, on peut vouloir représenter les items d'un type d'article dans la méta-base de données de la façon suivante. Seuls les items d'un premier niveau sont rattachés au type d'article par le méta-chemin d'accès «type d'article vers item». Pour représenter les autre niveaux dans les items, on a un méta-chemin d'accès «item vers item», qui est tel que son origine est l'item composé, et les cibles sont les items imposants. Comme à priori on ne peut pas savoir combien de niveaux d'items sont utilisés, il faudrait pouvoir boucler récursivement sur le méta-chemin d'accès «item vers item» jusqu'à épuisement d'une branche. Nous n'avons pas prévu cette possibilité, mais elle pourrait faire l'objet d'une extension future. Le problème des items a été résolu par le fait que la représentation de la structure des items dans la méta-base de données n'est pas celle supposée ici. En fait, dans la méta-base de données, tous les items sont direc-

tement rattachés au type d'article. La hiérarchie est retrouvée par l'ordre dans le méta-chemin d'accès et par le numéro de niveau que la description de chaque item comprend.

Si l'on travaille sur des boucles, il peut être parfois intéressant de sortir d'une façon prématurée d'une ou plusieurs boucles. Ainsi, on s'est donné une directive de sortie de boucle (fig. 5.1.4.).

break nombre de boucles

fig. 5.1.4. – sortie de boucle

Cette directive permet la sortie d'une ou plusieurs boucles selon le nombre indiqué.

b) Sélection

Selon les valeurs des items des méta-objets, il est parfois nécessaire de générer une partie de texte, alors que selon d'autres valeurs de cet item, il ne faut pas le générer. Il faut donc se donner une possibilité de sélection (fig. 5.1.5.).

if condition **then** < texte > **fi**

fig. 5.1.5. – sélection «si – alors»

Il serait bien entendu intéressant d'avoir une structure de sélection plus complète (fig. 5.1.6.).

if condition **then** < texte 1 > **else** < texte2 > **fi**

fig. 5.1.6. – sélection «si – alors – sinon»

Cependant, nous n'avons retenu que le premier format car il nous semble amplement suffisant, même si ce dernier est plus facile à l'usage. Ce deuxième format pourrait faire l'objet d'une extension future.

Comme la façon dont une interface doit être constituée est souvent très compliquée, il est intéressant de se donner la possibilité d'imbriquer des sélections.

Quelles possibilités donner à la définition de la condition ? La possibilité

d'imbrication de sélections nous permettrait d'accepter seulement des conditions élémentaires (fig. 5.1.7.).

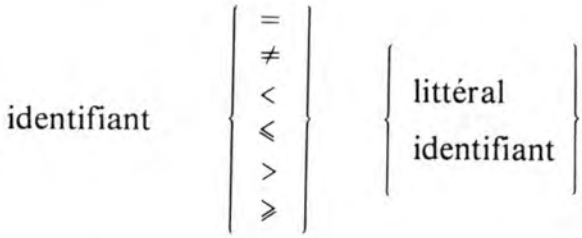


fig. 5.1.7. – format des conditions élémentaires

En faisant bien les imbrications, on peut généralement arriver au même résultat qu'avec des conditions plus complexes.

Ainsi par exemple, on peut traduire :

if (A = a ∧ B ≠ b) ∨ C = c **then** < texte 1 > **fi**

par :

```
if C ≠ c then
        < texte 1 >
fi
if C = c then
  if A = a then
    if B ≠ b then
          < texte 1 >
    fi
  fi
fi
```

Comme le montre cet exemple simple, la restriction à une condition élémentaire entraîne d'une part une traduction non-triviale et d'autre part une multiplication de texte.

On admet donc des conditions plus complexes dans lesquelles les conditions élémentaires peuvent être combinées par des ET, OU et NON logiques, ainsi que par une structure de parenthèses.

c) Garnissage des variables de gestion

Lorsqu'on parcourt une boucle d'accès, il est parfois nécessaire de connaître encore certaines informations sur un méta-article accédé avant le méta-article courant. Ainsi lorsque tous les méta-articles d'un type définissent une même zone dans un texte COBOL, il faudra pouvoir générer :

01 méta-article-n REDEFINES méta-article-n-1

Il faut donc pouvoir mémoriser l'identification du méta-article précédent. Afin de pouvoir mémoriser certaines informations, le générateur met à la disposition des macros un certain nombre de variables numériques et alphanumériques qu'on appellera variables de gestion.

Pour garnir ces variables de gestion, on se donne une directive de garnissage (fig. 5.1.8.).

move { littéral / identifiant } **to** variable de gestion

fig. 5.1.8. – directive de garnissage

Ici l'identifiant peut désigner n'importe quelle variable, donc aussi une variable de gestion.

Les variables de gestion numériques ne peuvent recevoir que des valeurs entières.

d) Calcul

Parfois on ne dispose pas directement d'une valeur à générer, mais il faut la calculer. Ainsi, si nous reprenons l'exemple des méta-articles qui redéfinissent une même zone, il est probable que les longueurs de ces méta-articles ne sont pas les mêmes et donc, après avoir les items du méta-article, il faut insérer un FILLER pour atteindre la bonne longueur. Le PICTURE de ce FILLER doit être calculé.

Le résultat d'un calcul ne peut aller que dans une variable de gestion numérique.

Comme type d'opération, nous n'admettons que l'addition et la soustraction d'entiers, car ce sont les seules dont nous avons ressenti l'intérêt.

add { littéral / identifiant } **to** { littéral / identifiant } **giving** variable de gestion numérique

subtract { littéral / identifiant } **from** littéral / identifiant **giving** variable de gestion numérique

fig. 5.1.9. – directive de calcul.

Avec les directives de calcul et la directive de garnissage, on peut utiliser une variable de gestion numérique comme compteur. Cette variable est initialisée par la directive de garnissage et elle est incrémentée/décrémentée par une directive de calcul.

e) Début/fin

On veut s'assurer que le générateur lit bien tout le texte des macros. Pour cela, on lui donne la possibilité de tester s'il a commencé par le début et atteint la fin. Ainsi on impose que tout le texte de commande de génération commence par une directive de début et se termine par une directive de fin. Bien sûr ceci ne permet pas de détecter s'il n'y a pas une partie de texte qui a été perdue entre le début et la fin.

5.1.3. Les commandes de génération

La commande de génération correspond à une partie de texte qu'il s'agit de recopier dans le texte en sortie, à la suite du texte déjà généré, après avoir fait les remplacements des paramètres par leurs valeurs courantes. Le problème principal qui se pose ici est le problème du remplacement des paramètres. Pour résoudre ce problème, il faut répondre aux questions suivantes :

- que doit-on entendre par paramètre à remplacer ?
- comment détecter qu'il y a lieu de faire du remplacement ?
- comment trouver la bonne valeur ?
- comment insérer la bonne valeur dans le texte à générer ?

Par paramètre, nous entendons ici toutes les références à des valeurs connues par le générateur et que les macros peuvent référencer afin de faire une sélection, un calcul, un garnissage de variable de gestion ou d'opérer un remplacement de texte.

Un premier type de paramètres sont les items des méta-articles que le générateur accède, suite à l'exécution d'une directive de boucle. Ces items ne peuvent être référencés dans les macros en dehors de la boucle qui accède à ce méta-article. Si à un méta-article sont rattachés des particularités, celles-ci sont référençables dans les mêmes conditions que les autres données du paramètre courant.

Un deuxième type de paramètres correspond aux variables de gestion. Ces variables servent au sauvetage, au calcul, et peuvent jouer le rôle de compteur. Ces variables peuvent être référencées n'importe où dans le programme. La personne qui rédige les macros doit s'assurer que ces variables sont garnies avant de les consulter.

Afin de pouvoir détecter qu'une variable doit être remplacée, elle doit être précédée dans le texte initial par un signe spécial. Bien sûr, ne peut être utilisé pour le remplacement qu'un jeu de paramètres prédéfini et connu par le générateur.

La bonne valeur de remplacement est retrouvée par le générateur. Pour les variables de gestion, le problème est assez facile car les valeurs sont connues à tout moment par le générateur. Pour les variables liées à la méta-base de données, il faut que cette variable réfère une valeur faisant partie d'un méta-article, accédé dans une des boucles courantes. Cependant, il se peut qu'on ait accédé, sur des niveaux de boucles successifs, des occurrences différentes d'un même type de méta-article. Alors on a plus d'une valeur courante du type de variable à remplacer. Pour savoir quelle valeur prendre, on peut faire suivre le paramètre à remplacer par un nombre. Ce nombre, qui est entouré de deux signes spéciaux pour le délimiter, indique qu'il faut prendre la valeur accédée dans la n^e boucle à partir de l'intérieur qui a accédé le type correspondant. Par défaut, on suppose qu'il s'agit de la première boucle à partir de l'intérieur qui y a accédé.

Enfin, on veut savoir comment se présente la ligne générée après le remplacement. Si nous avons sur les macros à l'entrée :

Or, la valeur n'a pas toujours la même longueur et en particulier pas la longueur de # PARAMETRE [n]. Cependant, la valeur peut faire partie d'un mot plus long et il est donc indispensable qu'il y ait ajustement de texte, non seulement si la valeur est plus longue que # VAR [n], mais aussi si elle est plus courte.

Comme il s'agit avant tout de générer des textes COBOL, il s'agit de respecter la logique des lignes COBOL. En général, le texte en sortie respecte le même ordre de ligne que celui du texte en entrée. Cependant, le remplacement peut provoquer un allongement d'une ligne et poser des problèmes de débordement. Chaque fois qu'il y a un débordement, l'utilisateur est averti. Le générateur a lui-même une gestion de débordement, mais celle-ci ne fonctionne pas dans tous les cas (en particulier les coupures des littéraux ne se font pas bien). L'utilisateur est donc prié de vérifier que le débordement a été bien fait.

5.1.4. Commentaires

Il faut voir que les commentaires peuvent apparaître à deux niveaux. D'abord, il y a le commentaire qui commente aussi bien les macros que le texte généré. Ce texte est à traiter comme toute autre commande de génération. Ensuite il y a le commentaire qu'on veut faire apparaître dans les commandes et non dans l'interface générée. On se donne par conséquent le moyen de distinguer le texte-commentaire du reste du texte des commandes.

5.2. LE GENERATEUR

Dans un premier temps, on expliquera comment le générateur arrive à produire une interface en interprétant un texte de commande de génération. Ensuite on verra quelles sont les demandes de génération à la méta-base de données et à quels moments ces requêtes sont lancées.

5.2.1. Le générateur et les commandes

Le générateur travaille en interpréteur des macros fournies en entrée, ce qui veut dire qu'il les lit et puis les exécute. Il espère travailler sur un texte correct, ce qui veut dire qu'il travaille en une seule passe et qu'il ne fait donc pas une vérification préalable de l'exactitude du texte soumis. La génération est commencée dès que le générateur a trouvé la directive de début sur le fichier des macros. L'apparition de la directive de début déclenche une demande à l'utilisateur du nom du sous-schéma et l'ouverture de la méta-base de données et du sous-schéma. Puis le générateur continue à interpréter le texte en entrée jusqu'au moment où il rencontre la directive de fin ou une erreur dans les macros. Si l'un des deux est rencontré, le générateur ferme le sous-schéma et la méta-base de données et il avertit l'utilisateur que la génération s'est terminée, soit normalement, soit anormalement, en indiquant dans ce dernier cas le type d'erreur rencontrée ainsi que l'endroit où cette erreur a été reconnue.

Cette approche de travail en une seule passe est critiquable. En fait, il semble être plus intéressant de travailler en deux temps. D'abord, on parcourrait le texte des macros pour détecter toutes les erreurs qui peuvent être détectées à priori, puis seulement on procéderait à la génération. Cette méthode présente plusieurs avantages. D'abord, lors de la mise au point des macros, il faut veiller à avoir un sous-schéma de test qui permet d'exécuter tous les corps de boucle et de sélection, afin d'assurer qu'on n'a pas d'erreurs dans ces parties des macros. Ainsi on diminuerait donc les risques de terminaisons anormales. Ensuite, on peut espérer avoir de meilleures performances lors de la génération car on peut supposer que

les macros sont correctes et il ne faudra donc pas faire autant de tests. Cependant, pour limiter l'ampleur de notre travail, nous avons choisi de procéder en une seule passe, tout en pensant qu'on n'a pas retenu la solution optimale du point de vue des performances.

Pour gérer les boucles, le générateur dispose d'une pile qui permet d'enregistrer d'une part toutes les informations nécessaires pour pouvoir faire appel à la méta-base de données et, d'autre part, les réponses de la méta-base de données. Chaque fois que le générateur rencontre une directive de début de boucle, il enregistre le type de boucle et, selon le cas, le type de méta-article, le type de méta-chemin d'accès, le type de méta-clé ainsi que la valeur de méta-clé. En plus, il enregistre le numéro de la ligne courante dans le fichier des macros. Si la réponse de la méta-base de données est positive, donc qu'on a trouvé l'article cherché, le corps de la boucle est exécuté; si non, il est sauté. A chaque fin de boucle, le générateur consulte la méta-base de données pour savoir s'il faut encore une fois exécuter la boucle ou non. Pour exécuter encore une fois une boucle, le générateur se resitue au début du corps de la boucle à l'aide du numéro de ligne enregistré. On voit donc qu'une commande sur le fichier des macros est lu autant de fois qu'elle est exécutée par le générateur. Ceci peut sembler accroître d'une façon importante le nombre de lectures sur le ficher des macros. Mais comme on ne peut pas prévoir une taille maximale pour le corps de la boucle, et donc qu'on ne peut pas le garder facilement en mémoire centrale, la solution qui consiste à tout relire à chaque fois a été retenue. Cependant, comme souvent le corps des boucles, ou au moins celui des boucles les plus intérieures s'il y en a plusieurs, est assez petit, on peut espérer limiter le nombre des lectures en choisissant des tampons assez importants.

Lorsque le générateur rencontre une demande pour sortie de n boucles, il va se positionner sur le fichier des macros derrière la fin de boucle correspondant aux n sorties, tout en mettant la pile de gestion de boucle à jour.

La sélection est traitée en deux temps. D'abord il y a une analyse et une évaluation de la condition selon les règles définies. Pour évaluer la condition, le générateur se sert de deux piles : d'une part d'une pile des résultats partiels obtenus et, d'autre part, d'une pile des opérateurs (NOT, AND, OR, ( ) non encore appliqués. Ensuite, selon le résultat de l'évaluation de la condition, le corps de la sélection est exécuté ou non.

Le générateur doit assurer qu'une sélection soit entièrement comprise dans une boucle ou inversément.Il ne peut donc pas y avoir chevauchement d'une boucle et d'une sélection. Pour pouvoir vérifier que les imbrications se font bien, le générateur dispose d'une pile qui comptabilise toutes les boucles et sélections non-encore terminées.

Les directives de garnissage et de calcul ne posent pas de problèmes majeurs, sauf que le générateur doit assurer que le type des opérantes et des résultats soient compatibles.

Lorsque le générateur rencontre une commande de génération, il recopie le texte en entier sur le fichier en sortie à l'endroit courant, tout en respectant la dispositon en lignes du texte original. Quand le générateur rencontre le signe spécial indiquant qu'il faut procéder à un remplacement, la recopie est interrompue. A l'aide de l'identification détectée derrière le signe spécial, le générateur peut savoir dans laquelle de ses zones mémoires il trouvera l'information à insérer dans le texte. Il la recopie à l'endroit courant de la ligne courante. Puis il continue le traitement de la commande de génération en faisant l'ajustement sur la ligne. Si c'est nécessaire, il fera un traitement de débordement en fin de ligne.

5.2.2. Le générateur et la méta-base de données.

Le générateur demande l'ouverture en consultation de la méta-base de données et d'un sous-schéma lorsqu'il commence la génération. Ils sont gardés ouverts pendant toute la génération jusqu'au moment où celle-ci se termine, normalement ou anormalement.

Avant d'exécuter le corps d'une boucle, le générateur lance chaque fois une requête à la méta-base de données pour avoir accès à un méta-article. La première fois, avant d'entrer dans la boucle, il demande l'accès au premier méta-article répondant aux critères, puis à chaque fois l'accès au méta-article suivant, en donnant la référence du précédent. Pour cette raison, le générateur garde dans la pile de gestion de boucle la référence des méta-articles courants. Lorsque tous les méta-articles correspondant aux critères sont parcourus, la réponse de la méta-base de données sera négative à la prochaine requête. A ce moment, le corps de la boucle n'est plus exécuté mais le générateur passera au texte de commande suivant la boucle.

Les trois formats de la boucle correspondent à trois types différents de consultation de la méta-base de données.

Le premier format,

for-each type de méta-article **do** < texte > **od**

est traduit par le générateur dans une demande d'accès à toutes les occurrences d'un type de méta-articles dans le sous-schéma ouvert.

Le deuxième format

for-each type de méta-article **within** type de méta-chemin d'accès **do** < texte > **od**

est traduit par une demande d'accès à toutes les cibles du type de méta-article dans le méta-chemin d'accès pour le sous-schéma ouvert. Le méta-chemin d'accès est déterminé par son type et par l'occurrence de l'origine dans une des boucles extérieures. Si plusieurs origines sont possibles dans différentes boucles extérieures, on considérera toujours la plus intérieure.

Le troisième format

for-each type de méta-article **using** type de méta-clé

equal { littéral / identifiant } **do** < texte > **od**

est traduit par une demande d'accès à tous les méta-articles d'un type, pour le sous-schéma courant, qui vérifient une valeur de la méta-clé donnée.

5.3 POINT DE VUE CRITIQUE

Les solutions que nous avons adoptées ici peuvent être discutées du point de vue de l'implémentation du générateur.

Les possibilités du langage de génération peuvent sembler limitées. Cette limitation s'explique cependant facilement par la démarche que nous avons suivie : une primitive n'a été introduite que lorsque le besoin s'est fait sentir et lorsque la mise en oeuvre nous a semblé raisonnablement faisable. C'est d'ailleurs pour des raisons de complexité que nous avons finalement renoncé à mettre en oeuvre les boucles récursives. L'absence de cette possibilité de récursivité nous semble être la plus grande limitation du point de vue des possibilités du langage.

Lors de l'implémentation du générateur nous avons poursuivi davantage la facilité de mise en oeuvre que la recherche des meilleures performances. Ce choix peut se justifier par la complexité du problème. Cependant, comme on l' a déjà souligné, nous regrettons de n'avoir pas eu le temps de réaliser un programme qui vérifie l'exactitude de la syntaxe des macros afin que le générateur puisse travailler en faisant moins de vérification et donc plus rapidement.

DEUXIEME PARTIE :

APPLICATION A DBMS-20

CHAPITRE 6. DBMS-20

INTRODUCTION

Le SGBD du DECsystem-20 (DBMS-20) est un système CODASYL basé sur les spécifications du Data Base Task Group (1971). Ce chapitre est consacré à l'étude de la structure de données et des primitives de ce SGBD. Pour ce faire, nous avons procédé à une étude comparative des éléments du MAG et du modèle d'accès de DBMS-20.

Cette étude est basée sur le langage de description des données de DBMS-20 (DDL) et sur son langage de manipulation des données (DML).

Cet exposé étant loin d'être complet, nous proposons au lecteur désireux d'avoir de plus amples informations de consulter directement | D1.D2.H12 | .

6.1. LES STRUCTURES DE DONNEES

Bases de données

A ce concept, appelé DATA BASE, est associé un nom qui identifie une base de données parmi l'ensemble des bases de données disponibles. Une DATA BASE est décrite par son SCHEMA, identifié par son nom parmi tous les schémas déclarés.

SCHEMA NAME IS schema-name.

Fichiers

Une base de données est découpée en espaces (fichiers) qui peuvent contenir des articles et qui sont appelés AREA. Une AREA est identifiée par un nom parmi les AREAS de la base.

AREA NAME IS area-name.

Article (Type D')

Les articles (RECORD) sont répartis en type (RECORD TYPE) identifiés par un nom au sein d'une même base de données. Un identifiant interne (DATA BASE KEY ou DBK) est associé aux articles; son référentiel est la base de données. Les articles d'un type peuvent être contraints à n'appartenir qu'à certaines AREAS.

RECORD NAME IS record-type-name
WITHIN area-name-1 (area-name-2... AREA-ID IS identifier)

Article système

Il existe un concept DBMS-20 (appelé SYSTEM) qui correspond à cet article; il ne lui correspond aucun item; il ne peut être ni créé ni supprimé comme des RECORDS.

Item (Valeur d'item)

A un type d'articles peuvent être associés O, 1 ou plusieurs items. Un item porte un nom unique parmi tous les items de la base de données. Un item peut être élémentaire ou décomposable; il peut être simple ou répétitif; il est toujours obligatoire.

un item est déclaré comme constituant d'un type d'articles ou d'un item décomposable. Cette description se présente dans l'ordre dynastique de la structure de décomposition à la manière d'une déclaration COBOL.

level-number item-name (item-description) (repet.)

Chemin d'accès (Type de)

Les articles d'une base (SYSTEM compris) peuvent être structurés en SETs, correspondant à des chemins inter-articles 1-N munis de leur inverse. A un type de chemins 1-N correspond un SET-TYPE nommé; le type d'articles origine est désigné comme OWNER et le, ou les types de cibles sont désignés comme MEMBERs du SET TYPE.

SET NAME IS set-type-name

OWNER IS { record-type-name / SYSTEM }

MEMBER IS record-type-name { MANDATORY / OPTIONAL } { AUTOMATIC / MANUAL }

MEMBER IS ...

Un même type d'articles ne peut être OWNER et MEMBER du même SET TYPE; le SYSTEM ne peut être MEMBER d'un SET.

Le type de chemins 1-N d'un SET-TYPE peut avoir des types de cibles déclarés chacun automatiques (AUTOMATIC) ou manuels (MANUAL) ainsi que obligatoires (MANDATORY) ou facultatifs (OPTIONAL).

Clé d'accès

On peut distinguer quatre mécanismes définissant des clés d'accès dans les référentiels suivants :

- tous les articles de la base de données : cette clé est la DATA BASE KEY qui est en outre un identifiant interne;
- les articles d'un type dans une AREA : il s'agit de la clé CALC (identifiante ou non); il y en a au plus une par RECORD-TYPE :

  LOCATION MODE IS CALC USING item-name-1, item-name-2, ...
- tous les articles MEMBERs d'un SET : si le SET-TYPE a été déclaré trié (SORTED) :

  ORDER IS SORTED
- les articles d'un type, MEMBERs d'un SET : pour chaque type MEMBER, on peut demander une clé d'accès dans le SET :

  ASCENDING KEY IS item-name-1, item-name-2, ...

Identifiants

On distingue quatre types d'identifiants selon le référentiel retenu :

- tous les articles de la base de données : DATA BASE KEY
- les articles d'un type dans une AREA : le groupe d'items déclaré comme clé d'accès CALC peut aussi être déclaré identifiant :

  LOCATION MODE IS CALC USING item-name-1, item-name-2, ...
  DUPLICATES ARE NOT ALLOWED

- tous les articles MEMBERs d'un SET : la clé de tri d'un SET-TYPE trié peut être déclarée identifiante :

  ORDER IS SORTED DUPLICATES ARE NOT ALLOWED

- les articles d'un type, MEMBERs d'un SET :

  ASCENDING KEY IS item-name-1, item-name-2, ...
  DUPLICATES ARE NOT ALLOWED

Chemins d'accès implicites

Les primitives autorisent l'exploitation de deux types de chemins d'accès implicites offrant :

- l'accès à tous les articles d'une AREA,
- l'accès à tous les articles d'un type dans une AREA.

Ordre des articles cibles d'un chemin

■ SET-TYPE à un seul type de MEMBER :

- ordre trié par valeurs croissantes de DATA BASE KEY :

  ORDER IS SORTED BY DATA-BASE-KEY

- ordre chronologique (FIFO) :

  ORDER IS ALWAYS LAST

- ordre antichronologique (LIFO) :

  ORDER IS ALWAYS FIRST

- ordre trié selon valeurs croissantes (ou décroissantes) d'une clé de tri; si clé de tri non identifiante, les valeurs doubles sont rangées par ordre FIFO (ou LIFO) :

  ORDER IS SORTED...
  ASCENDING
  DESCENDING KEY IS item-name-1,..

  DUPLICATES ARE { FIRST / LAST }

- ordre défini par programme (insertion après ou avant un MEMBER déterminé) :

  ORDER IS ALWAYS { NEXT / PRIOR }

■ SET-TYPE à plusieurs types de MEMBERs :

- ordre des MEMBERs indépendant des types :
  - . par DBK : ORDER IS SORTED BY DATA-BASE-KEY
  - . chronologique (anti)- : ORDER IS ALWAYS LAST (ou FIRST)
  - . trié : ORDER IS SORTED DUPLICATES... ASCENDING... pour chaque type MEMBER
- ordre trié en majeur sur le nom du type de MEMBER et en mineur, pour chaque MEMBER :
  - . sur une clé de tri : ORDER IS SORTED... ASCENDING
  - . sur les valeurs de DBK : ORDER IS SORTED – pas d'ASCENDING
- ordre trié des articles d'un type, mais pas de corrélation entre articles de types différents, pour chaque MEMBER :
  - . par clé de tri : ORDER IS SORTED WITHIN RECORD-NAME... ASCENDING
  - . par valeurs de DBK : ORDER IS SORTED WITHIN RECORD-NAME pas d'ASCENDING

Ordre des articles associés à une clé d'accès

Les articles associés à une clé d'accès non identifiante sont accessibles dans l'ordre chronologique (FIFO) ou antichronologique (LIFO) lorsque la clé est liée à un SET. Dans le cas d'une clé CALC, l'ordre est indifférent :

DUPLICATES ARE LAST (ou FIRST)

6.2. LES PRIMITIVES

Les sous-schémas COBOL

Un programme d'application travaille sur une base de données vue non pas au travers de son SCHEMA mais plutôt de l'un de ses SOUS-SCHEMAS (schéma externe). Un sous-schéma reprend essentiellement un sous-ensemble du schéma.

Un sous-schéma est nommé :

SUB-SCHEMA NAME IS sub-schema-name.

Le sous-schéma reprend une ou plusieurs AREAS :

COPY area-name-1, ...
COPY ALL AREAS

Il prend 0, 1 ou plusieurs types de SETS :

COPY set-name-1,...
COPY ALL SETS

Il reprend un ou plusieurs types d'articles :

01 record-name
[02 item-name-1]
[...]
...
COPY ALL RECORDS

La description des items peut différer de celle du schéma, dans les limites de conversions COBOL. L'ordre des items peut différer de celui du schéma; certains items peuvent être absents; un nouvel item peut être décrit comme groupement d'items du schéma.

Influence d'un sous-schéma sur les primitives

Les seules primitives permises sur les données, à un programme d'application, sont :

- celles qui concernent les objets décrits dans le sous-schéma; cette règle est d'application non seulement pour les opérations explicitement demandées, mais aussi pour les opérations implicites qui peuvent en dériver au moment de l'exécution;
- parmi celles-ci, celles pour lesquelles toutes les contraintes d'intégrité déclarées dans le schéma peuvent être respectées au moyen des objets retenus dans le sous-schéma.

Les primitives et les ordres d'accès

Les ordres d'accès du DML COBOL permettent au programmeur de commander l'exécution de primitives d'accès aux données d'une base.

Nous classerons les primitives d'accès en :

– accès à une base de données	INVOKE
– accès à un fichier	OPEN, CLOSE
– accès à un fichier	FIND
– accès à des valeurs d'item	GET
– accès à la description d'un article	MOVE

Les ordres d'accès aux articles (FIND) ont pour effet principal de mettre à jour les courants conformément aux règles de gestion qui ont été définies plus haut.

Accès aux fichiers

Toute AREA dans laquelle un programme désire travailler doit faire l'objet d'une demande préalable d'accès; l'acceptation correspond à l'ouverture de l'AREA pour le programme :

OPEN AREA area-name-1, ... USAGE-MODE IS ...
OPEN ALL

En fin de travail, le programme doit déclarer qu'il cesse toute activité dans l'AREA et demande la fermeture de celle-ci; l'arrêt d'un programme entraîne la fermeture automatique des AREAS ouvertes :

CLOSE AREA area-name-1, ...
CLOSE ALL

Accès aux articles cibles d'un chemin

1er cas : **Types de chemins explicites (SET-TYPE)**

– Accès au premier (dernier) article (d'un type) membre d'un SET

$$\text{FIND} \left\{ \begin{matrix} \text{FIRST} \\ \text{LAST} \end{matrix} \right\} [\text{record-type-name}] \text{ RECORD OF set-t-name SET}$$

– Accès à l'article suivant (précédent) (d'un type) membre d'un SET :

$$\text{FIND} \left\{ \begin{matrix} \text{NEXT} \\ \text{PRIOR} \end{matrix} \right\} [\text{record-type-name}] \text{ RECORD OF set-t-name SET}$$

- Accès à l' ième article (d'un type) membre d'un SET :

FIND { integer / ident. } [record-type-name] RECORD OF set-t-name SET

2ème cas : **Types de chemins implicites**

- Accès au premier (dernier) article (d'un type) dans une AREA :

FIND { FIRST / LAST } [record-type-name] RECORD OF area-name AREA

- Accès à l'article suivant (précédent) (d'un type) dans une AREA :

FIND { NEXT / PRIOR } [record-type-name] RECORD OF area-name AREA

- Accès à l'ième article (d'un type) dans une AREA :

FIND { integer / ident. } [record-type-name] RECORD OF area-name AREA

3ème cas : **Chemin inverse d'un Set**

- Accès à l'article OWNER d'un SET :

FIND OWNER RECORD OF set-t-name SET

Accès par clé d'accès

- Accès à un article (d'un type) par la clé interne (DATA BASE KEY) :

  FIND [rec-t-name] USING id.

- Accès à un article d'un type par la clé CALC :

  . Accès au premier :

  FIND rec-t-name RECORD

  . Accès au suivant :

  FIND NEXT DUPLICATE WITHIN rec-t-name RECORD

- Accès par clé à un article d'un type membre d'un SET :

  . Accès au premier :

  FIND rec-t-name VIA CURRENT OF set-t-name USING it1, it2, ...

  . Accès au suivant : (idem)

Accès aux valeurs d'item d'un article

GET [rec-t-name]
GET rec-t-name item-name-1, item-name 2, ...

Les primitives de mise-à-jour

- Création d'un article d'un type :

STORE rec-t-name

- Suppression d'un article (d'untype) (ou des membres d'un article) :

DELETE [rec-t-name] { ONLY / SELECTIVE / ALL } [MEMBERS FROM ...]

- Modification de valeurs d'item associé à un article :

MODIFY [rec-t-name]
MODIFY [rec-t-name] it1, it2, ...

- Insertion d'un article (d'un type) dans des chemins :

INSERT [rec-t-name] INTO set-t-name-1, ...
INSERT [rec-t-name] INTO ALL SETS

- Retrait d'un article (d'un type) de chemins :

REMOVE [rec-t-name] FROM set-t-name-1, ...
REMOVE [rec-t-name] FROM ALL SETS

CHAPITRE 7. L'ANALYSEUR

Rappelons que l'analyseur est un programme qui a pour fonction de garnir la B.D. schéma à partir d'une description de schéma donnée. Un analyseur est spécifique à un SGBD donné. Dans notre cas, le SGBD est celui du DEC-20, DBMS-20, et la description du schéma est écrite dans un langage qui est le DDL du DBMS-20. Ce chapitre est destiné à donner les spécifications de cet analyseur et un bref aperçu de sa façon de procéder.

7.1. SPECIFICATIONS

7.1.1. Entrée

Comme donnée d'entrée, nous avons une description de schéma écrite dans le DDL de DBMS-20. La syntaxe et la sémantique de ce DDL se trouve dans [D2]. Pour éviter des traitements d'erreurs inutiles, nous avons supposé que cette description était correcte d'un point de vue syntaxique. Un moyen assez évident pour vérifier l'état correct d'une description étant de la compiler sans erreur. Cependant, une description de schéma ne pourra pas toujours être analysée telle quelle, ceci pour deux raisons :

1. Restrictions du DDL.

Faute de temps, nous avons été contraints de restreindre les clauses permises par le DDL. Ces restrictions sont énoncées en annexe (manuel d'utilisation de l'analyseur).

2. Codes « externes »

Les codes externes sont les codes utilisés dans les paramètres d'appel aux interfaces pour identifier des objets tels que un type d'articles, un item, un type de chemins ou une clé d'accès. Afin de donner une certaine indépendance aux programmes d'application vis-à-vis des interfaces, il est indispensable que ces codes puissent être fixés une fois pour toute, même si l'on en venait à changer la description du schéma (ex. : ajout d'un nouveau type d'articles, d'un item,...). Ceci peut être rendu possible en permettant aux utilisateurs d'imposer ces codes lors de l'analyse de la description du schéma. Le moyen le plus simple de tenir compte de cette attribution a été de l'incorporer à la description même du schéma. Les règles concernant l'insertion de l'attribution des codes externes dans la description du schéma sont données dans le manuel d'utilisation de l'analyseur (cfr annexe).

7.1.2. Sortie

En sortie, nous trouvons les fichiers constituant la B.D. schéma. Il s'agit des fichiers des sous-schémas, des méta-objets, des méta-chemins d'accès et des particularités. Le format de ces fichiers est donné dans la chapitre 4 consacré à l'étude de la B.D. schéma et de son SGBD.

7.1.3. Fonction

La fonction de l'analyseur est de garnir les fichiers constituant la B.D. schéma à partir d'une description correcte de schéma écrite dans le langage DDL de DBMS-20.

7.2. PRINCIPE DE FONCTIONNEMENT

7.2.1. Description du D.D.L.

Pour bien comprendre le principe de fonctionnement de l'analyseur, il est utile de donner une description des données d'entrée, c'est-à-dire du DDL et de textes écrits en DDL. Cette description sera basée sur celle qui a été donnée par J.P. Robert.

Le DDL est formé d'un ensemble de clauses, chacune d'elles pouvant également être décomposable en sous-clauses.

Une clause ou sous-clause est un ensemble de mots qui permet de définir un élément ou une propriété d'un élément d'un schéma. Une clause est identifiable

par un ou plusieurs mots qui la composent. Ces mots ou ensembles de mots sont appelés «sélecteurs». Certaines clauses doivent obligatoirement être terminées par un mot spécial de fin.

Les mots composant les clauses du texte DDL peuvent être classés de la façon suivante :

1. les éléments non terminaux
   1.1. les éléments qui participent à un «sélecteur» (obligatoires)
   1.2. les éléments facultatifs
   1.3. les éléments identifiant un élément d'un schéma (record-name, item-name,...).
2. les éléments terminaux
   1.1. les éléments terminaux obligatoires
   2.2. les éléments terminaux facultatifs
   2.3. les éléments terminaux identifiant un élément d'un schéma.

Les différentes clauses peuvent apparaître dans une description de schéma selon un certain ordre. Cet ordre peut être représenté sous forme de clauses représentées par les sommets.

Un texte DDL (description de schéma) sera donc représenté par un chemin particulier parcouru dans ce graphe dont la cible est toujours fixée (END SCHEMA.).

7.2.2. Analyse syntaxique

Adaptée à ces données, la partie proprement dite d'analyse de l'analyseur sera considérée comme un automate fini, lui-même composé d'un certain nombre d'automates finis, chacun d'eux destiné à analyser une clause de niveau élémentaire (clause qui n'apporte qu'une information élémentaire). Un automate sera déclenché lors de la lecture du «sélecteur» de la clause qu'il doit analyser. La lecture qui suivra l'exécution d'un automate devra toujours avoir pour résultat le «sélecteur» de la clause qui suit celle qui vient d'être analysée. Cette partie d'analyse est basée sur le principe de la plupart des analyseurs syntaxiques et son étude approfondie ne serait que de peu d'intérêt; c'est pourquoi nous n'en dirons pas plus à ce sujet.

7.2.3. Remarques concernant d'autres analyseurs

L'analyseur comprend d'autres parties qui elles ont posé un certain nombre de problèmes dont l'étude pourrait être intéressante, ne fût-ce que pour la conception d'autres analyseurs du même type. Ces problèmes ont été dûs essentiellement aux exigences suivantes :

a) tri alphabétique des tables des fichiers, types d'articles, types de chemins et des items;

b) attribution facultative des codes externes (cfr plus haut) par l'utilisateur;

c) réalisation des relations inter-objets par une table de correspondance entre codes «internes» pour les relations 1-N et par un item contenant un code «externe» pour les relations N-1. Les codes internes sont les numéros d'ordre des objets dans leur tableau respectif;

d) notion de sous-schéma.

Ces exigences prises séparément n'auraient pas posé trop de problèmes, ce n'est que leur coexistence qui a été à la source de toutes les difficultés.

Tout d'abord, concernant les trois premières exigences:

Etant donné le tri alphabétique des tables, qui ne peut être fait qu'à la fin de l'analyse, les codes internes (numéros d'ordre dans les tables) ne sont connus qu'à la fin. Même chose pour les codes externes : vu que leur attribution est facultative, ce n'est qu'en fin d'analyse que l'analyseur attribuera des codes externes aux objets qui n'en ont pas encore reçus. D'autre part, les différentes relations sont détectées au fur et à mesure de l'analyse. Ceci fait (exigence a, b, c) que dans un premier temps, ces relations ne peuvent être mémorisées sous une forme définitive.

La façon de procéder que nous avons adoptée pour résoudre ce problème est la suivante :

Les relations sont représentées dans un premier temps par des relations entre les identifiants des objets qu'elles associent. Ces identifiants sont : l'identificateur pour les objets qui en possèdent un, ou le code interne pour les autres (ceux qui ne seront pas triés). Après avoir trié les tables et attribué les codes externes aux objets n'en ayant pas encore, ces relations sont transformées en relations définitives. Ceci est réalisé en remplacant, dans les relations, les identificateurs par les codes internes des objets qui les identifient.

La notion de sous-schéma (exigence d) imposa de garder en mémoire centrale toutes les tables contenant la description du schéma (objets, relations et particularités), et ce jusqu'à la fin de l'analyse. A chaque table d'objets a été associé un vecteur qui permet d'indiquer les objets qui ont été retenus pour un sous-schéma. Une partie de ces vecteurs (fichiers, types d'articles et types de chemins) sont garnis par la description du sous-schéma (AREA SECTION, RECORD SECTION, SET SECTION). Les autres vecteurs sont eux garnis à partir des relations obtenues à partir de la description du schéma. Ceci se fait par un principe de dérivation simple : si l'origine de la relation a été retenue (vecteur positionné), alors on retient la cible (positionnement du vecteur). Etant donné que le schéma des accès logiques est constitué d'au plus deux niveaux hiérarchiques

en dessous de T-ARTICLE, FICHIER ou T-CHEMIN, cette dérivation doit se faire en deux fois. La seconde fois ne sert qu'à dériver les composants (à partir des relations (SIMPLE, COMPOSANT) et (ITEM, COMPOSANT)).

Un autre problème concernant les sous-schémas est qu'ils peuvent avoir des objets en commun (fichiers, types d'articles, types de chemins, items, ...). Cette propriété n'apparaît pas de manière évidente lorsqu'on regarde le schéma conceptuel de la B.D. schéma. Pour cette raison, nous avons décidé de créer pour chaque sous-schéma une description indépendante dans la B.D. schéma. Ainsi par exemple, si un fichier (AREA) appartient à deux sous-schémas d'un même schéma, il apparaîtra deux fois dans la B.D. schéma. Une des conséquences entraînée par cette décision est que pour chaque sous-schéma, il faut créer des tables de relations et d'objets séparées. Vu que les tables d'objets ne sont pas nécessairement les mêmes que celles relatives au schéma, les codes internes (numéro d'ordre) des différents objets ont changé, ce qui fait qu'il faut mettre à jour les relations relatives à un sous-schéma en fonction des codes internes qui lui correspondent.

7.2.4. Particularités

Rappelons que la «particularité» est une notion qui permet de tenir compte pour n'importe quel SGBD de caractéristiques qui ne peuvent être décrites avec la structure de données du MAG (cfr chapitre 3). Ces caractéristiques peuvent concerner tout élément de la structure de données du SGBD. Dans notre cas, seules les caractéristiques nécessaires pour la génération des interfaces ont été retenues :

1. Chemins inverses

En DBMS-20 tout comme en CODASYL, les chemins d'accès sont de connectivité 1-N et ont leur inverse implicite. L'accès à partir d'une cible d'un chemin 1-N à son origine est toujours possible (FIND OWNER) .Pour tenir compte de cette caractéristique, l'analyseur crée, pour toute déclaration de chemin (SET) deux chemins dans la B.D. schéma. Le deuxième chemin est l'inverse implicite de celui qui est déclaré. L'analyseur lui attribue un nom (I- set-name) et lui associe un pointeur vers une particularité qui contiendra « IMPL-INVERSE ».

2. Schéma et sous-schémas

Un sous-schéma appartient à un schéma. Cette propriété n'est pas exprimable dans le MAG. C'est pourquoi il a fallu également en tenir compte au moyen d'une particularité associée à chaque sous-schéma. Cette particularité est utilisée lors de la génération de la SCHEMA SECTION des interfaces (INVOKE SUB-SCHEMA sub-schema-name OF SCHEMA schema-name). La particularité contient simplement le nom du schéma auquel appartient un sous-schéma.

CONCLUSIONS ET PERSPECTIVES

Arrivés au terme de ce mémoire, nous pouvons tenter une brève évaluation de ce qui a été réalisé.

Rappelons tout d'abord que l'objectif de ce travail était semblable à celui de J.P. Robert et consistait à mettre en œuvre un système de génération automatique d'interface pour les bases de données gérées par un SGBD bien spécifique. Dans notre cas, le SGBD était celui du DEC-20 : DBMS-20, un SGBD du type CODASYL.

Suite à l'expérience de J.P. Robert (R1) et celle que nous avons eue dans le cadre de notre stage chez NCR (N1), nous avons constaté qu'il était possible de concevoir un système de génération valable pour tous les SGBD. En effet, en analysant les systèmes de génération, nous avons constaté que certaines parties restaient communes à tout système. Malgré la lourdeur et la complexité supplémentaire que cela pouvait imposer, nous avons opté pour la mise en œuvre d'un système général. Ce choix a été essentiellement déterminé par le fait qu'un tel système faciliterait fortement la réalisation de génération automatique d'interfaces pour d'autres SGBD.

Si l'on veut mesurer l'impact, sur le système, de modifications intervenant aux différents niveaux de généralisation énoncés au chapitre 3 (schéma, SGBD, modèle d'accès), on constatera ceci :

- tout le système (BD schéma, générateur) reste invariant vis-à-vis des SGBD; l'introduction d'un nouveau SGBD nécessite cependant la mise en œuvre d'un analyseur et de macros spécifiques à ce SGBD.
- tout le système de génération reste indépendant des schémas

- vu que le système a été conçu dans l'optique d'un seul modèle d'accès, le MAG, un changement de celui-ci aurait des répercussions sur l'ensemble du système.

Finalement, on constate que l'objectif réel de ce travail s'est quelque peu éloigné de l'objectif de départ, ou du moins est devenu relativement plus vaste que celui-ci. Sa complexité de réalisation nous a contraints à ne considérer notre premier objectif que comme un cas d'application à un SGBD spécifique (DBMS-20) de notre système de génération et à lui imposer certaines limites. Celles-ci sont principalement :

- pas de mise en œuvre des primitives de contrôle
- pas de gestion de la concurrence
- pas de points de reprises
- restrictions relatives à la définition des sous-schémas.

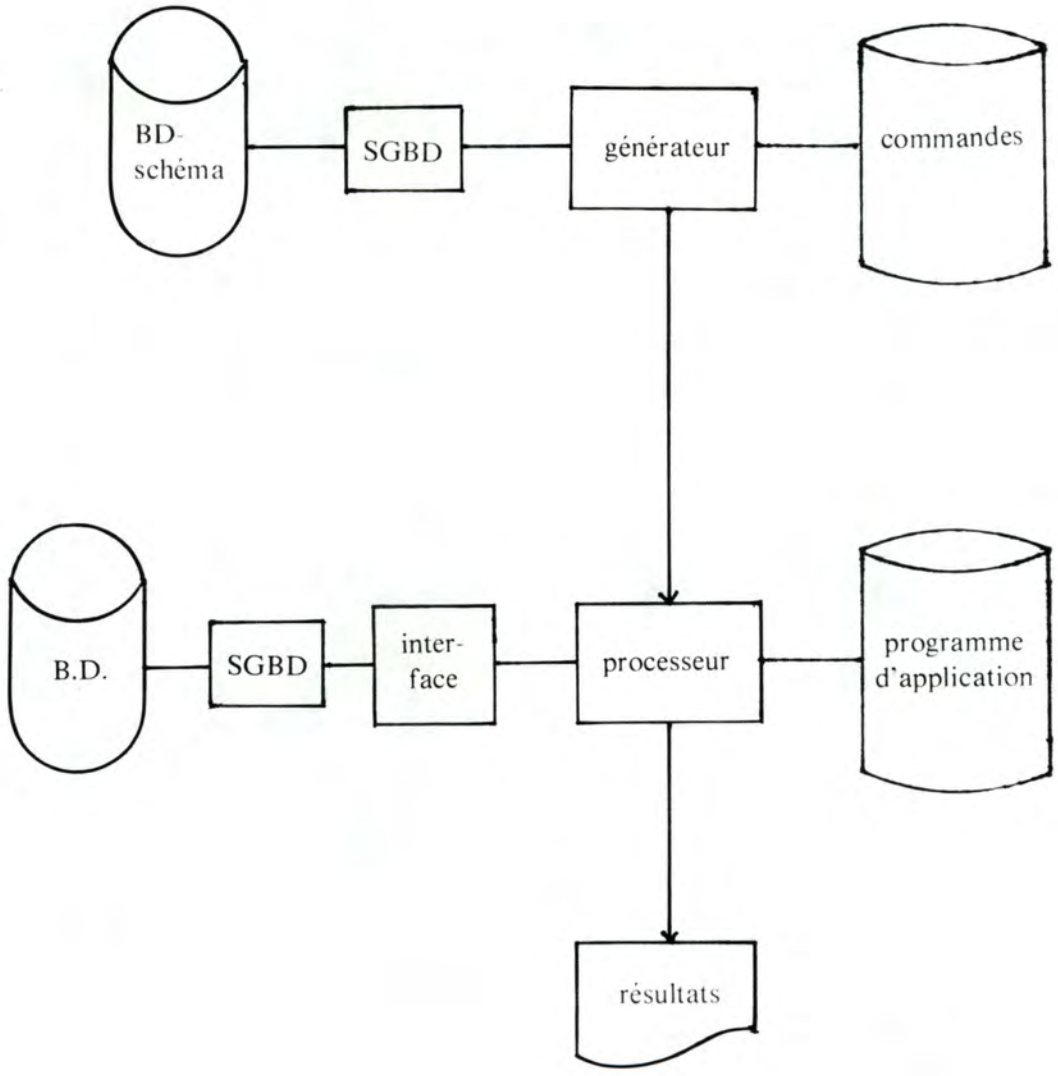
Si l'on regarde maintenant les difficultés auxquelles nous avons dû faire face, nous dirons que l'essentielle, outre le manque de temps évident, a été le test du système. En effet, le système est composé de plusieurs modules liés les uns aux autres, ne fût-ce que par leur structure de données; Ainsi, le test consistait en un test séparé pour chaque module et en un test d'intégration des différents modules. Les premiers tests ont posé pas mal de problèmes étant donné qu'ils exigeaient la simulation de leur environnement. Le test d'intégration quant à lui fut plus aisé.

Outre les nombreuses possibilités offertes par le système, nous pensons que, moyennant un minimum d'adaptation, il permettrait de réaliser sur n'importe quelle B.D. ce qui peut l'être actuellement sur la B.D. schémas. Ainsi, on pourrait envisager des applications telles que :

. la génération de rapports, statistiques sur le contenu d'une B.D.

. toutes sortes de programmes d'application simples travaillant sur une B.D. (ex. : génération des lettres de rappels cfr chapitre 3).

Le système serait basé sur celui proposé au chapitre 3 (générateur de générateur).



Dans un premier temps, le générateur génère un autre générateur appelé « processeur ». Ensuite, ce processeur sera capable d'exécuter des programmes d'applications faisant appel à n'importe quelle base de données, et ce, au moyen d'une interface qui lui est dédiée.

BIBLIOGRAPHIE

B 1 BODART, F.
Notes de cours (1. et 3. licence)
Institut d'Informatique, Namur

C 1 CODASYL DATA BASE TASK GROUP
April 1971 Report
ACM, New York

D 1 DIGITAL EQUIPMENT CORPORATION
DBMS-20 Programmer's Procedure Manual
DFC, Marlboro Massachussetts

D 2 DIGITAL EQUIPMENT CORPORATION
DBMS-20 Administrator's Manual
DEC, Marlboro Massachussetts

D 3 DELVAUX, Y.
IDML - Exemple de programme
Institut d'Informatique, Namur, janvier 1981

D 4 DATE, C.J.
An Introduction to Database Systems
Addison-Wesley, 1977

H 1 HAINAUT, J.L.
Un modèle descriptif de bases de données au niveau organique : le Modèle d'Accès
Institut d'Informatique, Namur, mars 1980

H 2 HAINAUT, J.L.
IDML - Machine virtuelle CODASYL
Institut d'Informatique, Namur, mars 1979

H 3 HAINAUT, J.L.
Etude de la Set Occurence Selection dans les rapports CODASYL
Institut d'Informatique, Namur, mars 1979

H 4 HAINAUT, J.L. - DELVAUX, Y.
Système portable de manipulation de bases de données hétérogènes
Institut d'Informatique, Namur, 1981

H 5 HAINAUT, J.L.
IDML - An Interface for Efficient Use of CODASYL Data Bases
Institut d'Informatique, Namur, mars 1979

H 6 HAINAUT, J.L.
General Model and Languages For Logical Data Description and Manipulation in DBMSs
ISO/TC97/SC5/WG5 N22, février 1982

H 7 HAINAUT, J.L.
Notes de cours (1. et 2. licence)
Institut d'Informatique, Namur

H 8 HAINAUT, J.L.
Current Trends for Data Bases :
Conception de bases de données - Un essai d'analyse des concepts et des méthodes
Institut d'Informatique, Namur, juin 1982

H 9 HAINAUT, J.L.
Construction de schémas conceptuels d'une base de données - Etude d'un cas
Institut d'Information, Namur, novembre 1980

H10 HAINAUT, J.L. - DELVAUX, Y.
Représentation des identifiants, ordres, clés d'accès dans le méta-schéma
Institut d'Informatique, Namur

H11 HAINAUT, J.L.
Base de données des schémas
Institut d'Informatique, Namur, 1981

H12 HAINAUT, J.L.
Analyse des accès proposés par le DBGT CODASYL (1971)
Institut d'Informatique, Namur, juin 1980

H13 HAINAUT, J.L.
Theoretical and Practical Tools for Data Base Design
7. International Conference on Very Large Data Bases, september 1981

N 1 NCR : MEURISSE, V. - BOCK, P. - DELVAL, M.
DBCI Project
NCR-ESC (European Support Center), Bruxelles, janvier 1982

R 1 ROBERT, J.P.
Génération automatique d'interfaces d'accès à des bases de données COBOL
Institut d'Informatique, Namur, 1981 (mémoire)

T 1 TAYLOR, R.W. - FRANK, R.L.
Centralized and Distributed Data Base Systems :
CODASYL Data-Base Management Systems
IEEE, New York, 1979

W 1 WEDEKIND, H.
System Independent Program Design with Data Bases
Technische Universität, Damstadt, février 1978

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

GENERATION AUTOMATIQUE D'INTERFACE D'ACCES A DES BASES DE DONNEES

Annexes

septembre 1982

Mémoire présenté par
P. Bock / M. Delval
en vue de l'obtention du titre de Licencié et Maître en Informatique.

RUE GRANDGAGNAGE, 21, B - 5000 NAMUR (BELGIUM)

COMPLEMENT A L'ANNEXE D'UTILISATION DES INTERFACES.

Ce complément concerne essentiellement les interfaces générés pour les bases de données gérées par DBMS-20.

1. Appel aux interfaces.

Les interfaces DBMS-20 possèdent deux points d'entrée. Ceci est du au fait qu'un certain nombre de macros sont générées lors de la compilation des interfaces juste après la clause PROCEDURE DIVISION. Ce sont des macros qui réalisent le lien avec la base de données (INVOKE). Ces macros ne doivent etre exécutées que lors de la primitive d'ouverture de la base de données. C'est la raison pour laquelle, pour tous les appels autres que l'ouverture B.D., il existe un autre point d'entrée (ENTRY POINT) dans l'interface. L'identificateur de ce second point d'entrée est celui de l'interface terminé par un '2'. Le format des appels aux interfaces sera donc le suivant :

- pour une demande d'ouverture de la B.D. :

  CALL 'interface-name' USING <parameters>

- pour toutes les autres primitives :

  CALL 'interfaces-name2' USING <parameters>

2. Format des paramètres d'appels

cfr annexe page 33 :

le format de RETCODE est le suivant :

```
02 RETCODE  DISPLAY-9.
   03 FNCODE              PIC 99.
   03 ERRCODE             PIC 99.
```

3. Codes d'erreurs

Comme nous venons de le voir, RETCODE est composé de deux éléments : FNCODE et ERRCODE. Dans le cas ou il s'agit d'une erreur identifiable par l'interface, ERRCODE prendra une des valeurs énumérées dans l'annexe à la page 34 et FNCODE contiendra le code de la primitive dans laquelle l'erreur a eu lieu. Si l'erreur n'est pas reconnaissable par l'interface, FNCODE aura la valeur 99 et ERRCODE contiendra un des 'exception condition codes' de DBMS-20 (cfr DBMS Programmer's Procedures Manual, Appendix B).

Codes d'erreurs supplémentaires :

73 : SETLST incorrect;
si la liste SETLST est incomplète ou si elle contient un élément incorrect.

74 : CURLST incorrect
si un des éléments de CURLST n'identife pas un chemin du type donné par l'élément correspondant dans SETLST.

AVANT-PROPOS

Les annexes du mémoire sont réparties en deux volumes. Le premier volume est princpalement consacré aux manuels d'utilisation des différents programmes qui ont été mis en oeuvre dans le cadre du mémoire. Le deuxième volume contient les programmes sources.

Comme les différentes parties qui composent ces annexes sont indépendantes, il ne nous a pas semblé opportun de donner une numérotation continue aux pages de ces annexes. Pour la même raison nous avons préféré donner la table des matières détaillée au début de chaque partie plutôt qu'au début de ce volume.

Le volume I des annexes comprend dans l'ordre:

- le manuel d'utilisation des interfaces
- le manuel d'utilisation de l'analyseur pour DBMS-20
- le manuel d'utilisation de la méta-base de données
- le manuel d'utilisation du générateur
- l'exemple PETITPAS: exemple de génération
- le guide des changements du système

Le deuxième volume des annexes comprendra dans l'ordre :

- le programme du générateur: GENER
- le progamme de l'interface d'accès de la méta-bd: METABD
- le programme d'analyse du DDL DBMS-20: ANALYS
- le programme de transformation du texte de génération: TRANSF
- le texte de génération pour DBMS-20: MACRO
- l'interface générée pour la base de données de PETITPAS sur DBMS-20: FINTER

MANUEL D'UTILISATION

DES INTERFACES

Juillet 1982

PLAN DE L'ANNEXE.

Introduction.

Les interfaces générés sont utilisés par des programmes d'application (cfr chapitre 2). Cette annexe donne la manière dont ces interfaces doivent etre utilisés. Ce manuel est en principe valable pour tous les interfaces générés, quelque soit le SGBD sous-jacent. Cependant, quelques limitations sont indiquées pour DBMS-20 (CODASYL).

Un interface est invoqué par un appel; un appel correspond à une demande d'exécution d'une primitive. Les paramètres et zones de transmission d'informations sont transmis au moment de l'appel. Pour des programmes d'application COBOL, ces appels auront la forme :

CALL 'interface-name' USING <parametres>

Avant de donner la descritpion précise des primitives et de leur paramètres d'appel, il nous a semblé bon de définir quelques notions qui reviennent fréquemment par la suite. Il s'agit des notions d'ouverture de fichiers et de base de données, de référence à un objet et de codes d'objet.

1. Ouverture de fichiers et de bases de données.

1.1. Ouvertures multiples.

On peut supposer que la séquence des opérations sur B.D. qu'un module utilisateur adoptera, sera le plus fréquemment:

- ouverture de la B.D.
- ouverture(s) des fichier(s)
- mise à jour / consultation des fichier(s)
- fermeture des fichier(s)
- fermeture B.D.

Afin de permettre la standardisation de ces modules, et d'éviter toute contrainte sur l'architecture qui les structure (imbrication, simultanéité) la possibilité d'ouverture multiple de la B.D. et des fichiers a été prévue dans les interfaces.

1.2. Ouverture des fichiers - mode automatique et manuel.

Problème posé.

Le problème abordé ici, concerne l'ouverture des fichiers de la base de données. Nous pouvons considérer, d'une manière générale, que nous aurons à satisfaire deux types d'utilisateurs : ceux qui désirent ignorer la notion de fichier, et ceux qui ne le désirent pas.

Solution partielle.

Pour satisfaire ces deux types d'utilisateurs, il avait été proposé ( ) d'accompagner l'ouverture de la base de données, d'un mode d'ouverture. Ce mode d'ouverture pouvant etre " MANUEL " ou " AUTOMATIQUE ".

Mode automatique (A): ouverture dynamique des fichiers par l'interface.

Ce mode consiste à laisser le controle des fichiers à l'interface. Ainsi, la notion de fichier est rendue transparente à l'utilisateur. Principe: si une opération d'accès tombe sur un fichier fermé, l'interface l'ouvre et recommence immédiatement l'opération interrompue.

Mode manuel (M): controle explicite des fichiers par l'utilisateur.

Dans ce cas, les ouvertures de fichier sont entièrement à charge de l'utilisateur. Ainsi, toute tentative d'accès à un article appartenant à un fichier fermé échouera, et un code d'erreur sera retourné à l'utilisateur.

Problèmes secondaires.

Cette solution nécéssite cependant des précisions supplémentaires.

1. Protections:

On peut se demander, lorsqu'on travaillera en mode (A), quel sera le mode d'ouverture des fichiers qui sera utilisé ?

2. Coexistence des deux modes :

Admet-on que les modules d'un meme programme utilisent des modes d'ouverture différents ? Si oui, quelles sont les conditions d'enchainement de ces modes ?

3. Fermeture :

Lorsqu'on ferme une base de données qui avait été ouverte avec un mode automatique, qu'en est-il des fichiers ouverts automatiquement ?

Solutions proposées.

1. Protections :

un certain nombre de solutions ont été envisagées:

1.1. Ouverture minimale:

le mode (A) ouvrirait les fichiers avec le mode d'ouverture minimum; dès lors, plusieurs questions se posent:
- quelle ouverture minimale choisir ?

ici se pose un problème de comparaison entre les différents modes d'ouverture; par exemple, pour Codasyl, il existe les modes suivants :

-	RETRIEVAL
PROTECTED	
EXCLUSIVE	UPDATE

quel mode d'ouverture est plus exigeant ou plus restrictif qu'un autre ?

il n'existe pas de réponse bien précise à cette question; pour notre part, nous avons donné des règles précises de comparaison entre les modes d'ouverture Codasyl (cfr).

- que faire si une primitive exige un mode d'ouverture plus restrictif que le mode minimum ?
  --> soit un ajustement automatique du mode par l'interface (on ferme et on réouvre le fichier avec le mode nécéssaire)
  --> soit un ajustement manuel par l'utilisateur lorsqu'il exige un mode plus que restrictif au mode minimum.

2.2. Ouverture maximale:

le mode (A) ouvrirait les fichiers avec le mode d'ouverture maximum; (en CODASYL : EXCLUSIVE UPDATE);
il apparait directement dans cette solution une restriction énorme en cas de concurrence. Ceci pourrait etre amélioré en permettant à l'utilisateur un ajustement manuel.
(s'il veut etre plus efficace point de vue de la concurrence, il ne demandera que ce qu'il lui faut)

La solution adoptée a été d'accompagner le mode d'ouverture (A) de la B.D., d'un mode de protection qui sera celui utilisé lors de toute ouverture automatique de fichier.

2. Coexistence des deux modes :

Pour garder le caractère indépendant de la décomposition d'un programme en module, la coexistence des deux modes parait indispensable. Cependant, elle ne peut se faire qu'en respectant un certains nombres de règles :

a) après un mode manuel:

- toute autre ouverture B.D. en mode (M) sera acceptée.

- toute ouverture B.D. en mode (A) sera refusée;
  cette restriction a été adoptée d'une part par simplification, et d'autre part en tenant compte du fait qu'en général, ce ne seront que des modules 'techniques' et donc qui se doivent efficaces, qui utiliseront ce mode;

- toute ouverture de fichier sera acceptée si :

. le fichier n'est pas encore ouvert
ou . le fichier est déjà ouvert, et la protection demandée est compatible avec celle avec laquelle le fichier est ouvert.

b) après un mode automatique:

- toute autre ouverture B.D. en mode (A) sera acceptée si la protection demandée est compatible avec celle du mode (A) de niveau supérieur.

- toute ouverture B.D. en mode (M) sera acceptée.

- toute ouverture fichier sera acceptée si:

. le fichier n'est pas encore ouvert
ou . le fichier est déjà ouvert, et la protection demandée est compatible avec celle avec laquelle le fichier est ouvert.

3. Fermeture :

La décision prise concernant ce problème est la suivante :

Lors de la fermeture d'une B.D. ouverte en mode (A), aucune fermeture de fichier n'est réalisée sauf s'il s'agit de la dernière fermeture de la B.D. (fermeture physique), auquel cas tous les fichiers ouverts sont fermés automatiquement.

Compatibilité:

Dans le cas d' un SGDB Codasyl, la définition de compatibilité entre deux demandes d'ouverture sera la suivante:

Considérons qu'une demande d'ouverture X, est composé de deux paramètres:

- son utilistation u(X) qui sera : RETRIEVAL ou UPDATE

- sa protection p(X) qui sera : - ou PROTECTED ou EXCLUSIVE

On dira qu'une demande d'ouverture Y, est compatible avec une demande d'ouverture X, si:

$u(Y) <= u(X)$ et $p(Y) <= p(X)$

avec I RETRIEVAL < UPDATE
I
I - < PROTECTED < EXCLUSIVE

Dans tous les autre cas, on considérera que la demande (Y) n'est pas compatible avec la demande (X).

2. Notion de référence à un objet.

La référence à un objet est un identifiant associé automatiquement par l'interface à tout objet auquel on a accédé avec succès, et qui permet de le désigner lors de toute demande ultérieure le concernant. Les objets susceptibles de se voir attribuer une référence sont les bases de données, les fichiers et les articles.

La référence B.D. est attribuée de facon différente des autres : l'interface s'attend à recevoir cette référence lors de le première demande d'ouverture de la B.D.; il ne l'attribue pas lui-meme. Cette référence, qui permet d'identifier une B.D. parmi d'autres, ne joue pleinement son role que dans un contexte multi B.D.

Toute référence disparait avec le processus durant l'exécution duquel elle a été attribuée.

3. Notion de code de type d'objets.

Deux possibilités de désignation ont été envisagées : soit le nom complet, soit un code.

Les deux possibilités ont été utlisées dans les interfaces : B.D. et fichiers seront désignés par leur nom tandis que types d'articles, types de chemins, items et clés d'accès le seront par un code. Ce choix a été déterminé par la fréquence d'utilisation de ces désignations.

Les codes peuvent etre soit attribués automatiquement par le système générateur, soit laissés au choix de l'administrateur de la base de données, qui disposera dans la description du schéma, des clauses lui permettant de fixer les codes des types d'objets (cfr analyseur (chapitre 7) et manuel d'utilisation de l'analyseur (annexe)).

4. Les paramètres d'appel - Leur signification et leur utilisation.

Les interfaces sont suceptibles de réaliser de nombreuses primitives; encore faut-il que leur soient communiquées les informations nécessaires à l'exécution de celles-ci. C'est de ces paramètres que nous entreprenons maintenant la description.

Nous pouvons distinguer cinq classes particulières de paramètres :

- Les paramètres précisant l'opération à effectuer et les objets sur lesquels elle s'applique;
  nom générique : Z-CODES.

- Les paramètres permettant le transfert de valeurs d'identifiants ou de valeurs d'objets de l'utilisateur vers l'interface;
  nom générique : Z-IDENT/Z-VALUES.

- Des paramètres particuliers prévus pour la transmission d'une liste de codes d'items;

nom générique : Z-ITEMS.

- Des paramétres utilisés pour la transmission des valeurs d'items auxquelles un accès a été demandé (de l'interface vers l'utilisateur); nom générique : Z-RESP.

- Des paramètres qui concernent les chemins d'accès; nom générique : Z-SETS.

Chacune de ces classes est composée d'un certain nombre de paramétres dont nous allons donner la signification.

4.1. Z-CODES : cette classe comprend :

- COP : code opération; un code opération a été attribué à chaque primitive; lors de tout appel, COP doit contenir l'un de ces codes; si tel n'est pas le cas, aucune primitive ne sera exécutée. Les valeurs de COP correspndantes à chacune des primitives sont données lors de leurs descriptions.

- SREF : référence du schéma (de base de données); cette référence qui identifie la B.D., est fournie à l'interface par l'utilisateur lors du premier appel (ouverture B.D.); SREF devra contenir cette meme référence lors de tous les appels successifs, sinon, aucune action ne sera faite par l'interface. Lors d'une ouverture autre que la première (ouvertures multiples), SREF sera garni par l'interface, avec la référence qui lui aura été fournie lors de la première ouverture B.D.

- COREC : code du type d'article; COREC doit contenir un code numérique attribué à un type d'article, lors de la demande d'exécution d'une primitive qui requiert parmi ses paramètres, une identification de type d'articles.

- RETCODE : code de retour; contiendra après l'exécution d'une requete par l'interface, le diagnostic de cette exécution. La valeur 0 signifiera une exécution normale. Des valeurs différentes de 0 signifieront la détection d'une anomalie (paramètres incorrects, requete impossible à satisfaire ...). Les valeurs attribuées et leur signification figurent en annexe ...

- COMOD : code de modalité; ce paramètre a deux interprétations possibles:

  a. mode d'ouverture de la B.D. (CFR annexe ...)
     COMOD peut prendre les valeurs 0 ou 1;

     - 0 signifie mode d'ouverture automatique; la notion de fichier est ainsi rendue transparente à l'utilisateur; les accès aux articles sont permis sans ouverture préalable du fichier.

     - 1 signifie mode d'ouverture manuel; on ne peut accéder à un article de la B.D. sans avoir ouvert le

fichier qui le contient.

b. mode de suppression d'un article;
COMOD peut prendre les valeurs 0 à 3, qui détermineront la fonction précise de la primitive de suppression;

- 0 : si l'article n'est origine d'aucun chemin non vide, l'enlever de tous les chemins dont il est cible puis, (1)
  l'effacer de son fichier.
- 1 : enlever l'article de tous les chemins dont il est cible;
  enlever des chemins dont il est origine, toutes les cibles optionnelles et appliquer (1) à toutes celles qui sont (2)
  obligatoires;
  effacer l'article de son fichier;
- 2 : enlever l'article de tous les chemins dont il est cible;
  enlever des chemins dont il est origine toutes les cibles optionnelles qui sont aussi cibles d'autres chemins; (3)
  pour ces memes chemins, appliquer (3) à toutes les autres cibles (obligatoires ainsi que les optionnelles qui ne sont pas membres d'un autre chemin);
  effacer l'article de son fichier;
- 3 : enlever l'article de tous les chemins dont il est cible;
  appliquer (4) à toutes les cibles des chemins dont il est (4)
  origine;
  effacer l'article de son fichier;

- PROTECT : code de protection de fichier;
ce paramètre est utilisé dans deux cas:

a. lors de l'ouverture d'une base de données en mode automatique;
il précise le mode d'ouverture qui sera utilisé pour toute ouverture automatique de fichier réalisée par l'interface.

b. lors de l'ouverture d'un fichier;
il précise, lors d'une ouverture de fichier, le mode d'ouverture désiré par l'utilisateur pour ce fichier.

Le mode d'ouverture d'un fichier étant constitué de deux paramètres :

- la protection : aucune, protégée ou exclusive
- l'utilisation : consultation ou mise à jour

PROTECT pourra prendre les valeurs suivantes :

1 : consultation
2 : mise à jour
3 : consultation protégée
4 : mise à jour protégée
5 : consultation exclusive
6 : mise à jour exclusive

- COGET : code d'accès aux valeurs d'items d'un article;
ce paramètre permet lors d'un accès à un article de signaler à l'interface si l'on désire accéder aux valeurs d'item de l'article;
2 valeurs sont prévues :
0 : pas d'accès aux valeurs d'items

1 : accès à toutes les valeurs d'items

- CONTRL : controle d'article;
  permet de préciser le controle que l'interface doit exercer sur l'article auquel on accède. Une étude ultérieure de ces mécanismes généralisée à d'autres modèles de données, permettra de préciser davantage le role de ce paramètre.

- RFIL : référence de fichier;
  destiné à recevoir une référence de fichier lors de l'appel des primitives qui exigent une telle référence;
  RFIL est garni par l'interface lors de l'ouverture du fichier;
  toute requete ultérieure concernant ce fichier devra fournir cette référence.

- RREF : référence d'article;
  après un accès réussi à un article, RREF contiendra la référence de cet article; RREF devra etre garni par l'utilisateur pour l'appel des primitives qui requièrent une telle référence.

- ORDER : ordre d'accès;
  précise l'ordre dans lequel on désire accéder aux articles;
  valeurs possibles :

  0 : ordre naturel de création des articles;
  cet ordre peut etre significatif (FIFO, LIFO, TRIE, ...) ou non (ex. par clé CALC en CODASYL).
  1 : ordre trié selon les valeurs d'une clé de tri dont le code doit etre donné par COSIMPLE.

- POSIT : positionnement d'un article;
  permet de préciser la position d'un article que l'on veut accéder;
  dans tous les cas, cette position n'est significative que par rapport à une autre position relative (PREF) et à un ordre donné (ORDER).
  valeurs possibles :

  0 : l'article à accéder doit etre le suivant de l'article dont la référence est contenue dans PREF.
  1 : l'article à accéder doit etre le précédent de l'article dont la référence est contenue dans PREF.
  2 : l'article à accéder doit etre le ième article suivant l'article dont la référence est contenue dans PREF;
  la valeur de i doit etre contenue dans ZVALUE.
  3 : l'article à accéder doit etre le ième article précédant l'article dont la référence est contenue dans PREF;
  la valeur de i doit etre contenue dans ZVALUE.

- PREF : position relative d'un article;
  contient la référence (cfr RREF) d'un article qui servira de base pour accéder à un autre article; valeurs possibles :

  0 : ne correspond à aucune référence d'article;
  ainsi, selon les valeurs de POSIT, on accédera au premier (POSIT = 0 ou 2) ou au dernier (POSIT = 1 ou 3) dans l'ordre spécifié par ORDER.

  <> 0 : correspond à la référence de l'article qui indique la position

relative.

- COSIMPLE : code de groupe de controle;
définition de groupe de controle (SIMPLE):

  - un "item de controle" sera tout item associé à

    . un ordre et/ou
    . une clé d'accès et/ou
    . un identifiant

  - tout item n'ayant pas cette caractéristique sera appelé "item indépendant".

  - un "groupe de controle" (SIMPLE) est un ensemble d'items de controle qui correspondra selon les cas à un ordre et/ou une clé d'accès et/ou un identifiant.

COSIMPLE doit contenir le code d'un groupe de controle lors de l'appel des primitives requérant un groupe de controle; il s'agit des primitives telles que les accès par clé ou séquentiel dans un ordre trié et les modifications de valeurs d'items de controle.

- OPERAT : opérateur;
permet, associé à Z-VALIT, de préciser une condition sur la clé de l'article auquel on accède par clé. La condition est :

" clé de l'article accédé <OPERAT> <valeur de clé contenue dans Z-VALIT> "

valeurs possibles :

0 : pas de condition
1 : "="
2 : ">"
3 : "not <"

Seules les valeurs 0 et 1 sont reconnues par les interfaces DBMS-20.

- COSET : code d'un type de chemin;
contient le code d'un type de chemins d'accès inter-articles, lors de l'appel des primitives qui en exigent un.

- OREF : référence d'un article origine d'un chemin;
contient la référence de l'article origine d'un chemin d'accès lors de l'appel des primitives faisant intervenir des mécanismes d'accès inter-articles.

- TYP : en réserve;

5.2. Z-IDENT/Z-VALUE : cette classe comprend :

- SSNAME : nom de la base de données;
ce nom est à fournir lors de la demande d'ouverture de la B.D.
Il doit contenir le nom de la B.D. gérée par l'interface; dans le cas

contraire, l'accès à la B.D. sera refusé.

- PSW : mot de passe;
  mot de passe à fournir lors de la demande d'ouverture de la B.D. S'il ne correspond pas à celui défini pour la B.D. gérée par l'interface, aucun accès ne sera permis.

- FILENAME : nom de fichier;
  nom à fournir lors de l'ouverture d'un fichier; ce nom doit etre celui de l'un des fichiers appartenant à la B.D. gérée par l'interface.

- Z-VALIT : valeur d'item d'un article;
  destiné à contenir les valeurs d'items des articles en cas de création d'un article ou de modification de valeurs d'items d'un article; c'est ce meme paramètre qui contiendra les valeurs des items composant une clé d'accès, lors de l'utilisation d'une primitive d'accès par clé.

5.3. Z-ITEM.

- cette liste est destinée à définir le contenu de Z-VALIT et Z-RESP; les codes d'items correspondent successivement aux items dont les valeurs sont présentes dans Z-VALIT ou dans Z-RESP. Cependant, cette zone n'est pas utilisée par les interfaces DBMS-20 car cela exigait une manipulation des zones de données qui ne peut etre rendue par le langage COBOL (dans lequel sont écrits les interfaces DBMS-20). Ainsi, dans notre cas, la structure de décompostion de Z-VALIT sera toujours la meme pour un type d'article déterminé, càd que chaque item d'un article aura toujours une sous-zone fixe de Z-VALIT qui lui sera attribuée. Cette containte est valable aussi bien pour la création d'un article, que pour la modification des valeurs d'items ou l'accès par clé.

5.4. RFIELD.

- RFIELD est destiné à permettre le transfert de l'interface vers l'utilisateur, des valeurs auxquelles celui-ci accède.

5.5. Z-SETS.

- STKREF : référence de stockage;
  destiné à recevoir un paramètre de stockage physique, nécessité par certains SGBD, tels que CODASYL.

- SETLST : liste de codes de types de chemins d'accès;
  cette liste contient les codes des types des chemins dans lesquels un article doit etre inséré automatiquement lors de sa création; les références des articles cibles ou origines permettant d'identifier

ces chemins sont contenues dans la liste des références CURLST.

- CURLST : liste de références d'articles; contient les références des articles qui permettront d'identifier les chemins dans lesquels un article doit etre inséré automatiquement lors de sa création; aucun ordre n'est exigé sur le placement des éléements des listes SETLST et CURLST si ce n'est qu'il doit etre le meme pour ces deux listes.

5. Les primitives.

Nous abordons la spécification détaillée de chacune des primitives qu'un interface est susceptible de mettre à la disposition des utilisateurs. Tous les SGBD/SGF ne les offrent pas toutes; elles seront cependant toutes spécifiées, et pour certaines d'entre elles des restrictions seront signalées concernant les interfaces DBMS-20 (CODASYL).

Conventions.

Pour chacune des primitives, nous donnons dans l'ordre : les paramètres d'entrée et de sortie, et la fonction. Nous indiquons les diagnostics d'erreurs susceptibles d'etre transmis à l'utilisateur par l'interface (RETCODE).

Certains diagnostics, établis à l'entrée de l'interface, sont communs à toutes les primitives.

- " SREF invalide " si SREF ne contient pas la référence de la base de données qui a été fournie à l'interface lors de la première ouverture de la B.D. Ce diagnostic est valable pour toutes les primitives sauf "ouverture de la B.D. ".

- " B.D. non ouverte " si un appel autre que "ouverture B.D. est transmis alors que la B.D. est fermée.

- " primitive non implémentée " si COP contient le code d'une primitive non implémentée par l'interface appelé.

- " opération non définie " si COP ne contient pas le code d'une primitive.

5.1. Les primitives d'accès.

Les principaux objets à accéder sont la base de données, le fichier, l'article et la valeur d'item.

5.1.1. Accès à la base de données.

- Ouverture de la base de données.

entrée :

COP : 11
SSNAME
PSW
COMOD
PROTECT
SREF

sortie :

SREF
RETCODE

fonction :

- rattacher un utilisateur (programme) à la base de données s'il y est autorisé; cette primitive doit etre exécutée avec succès, au moins une fois avant toute autre primitive.

- vérifier l'existence de la B.D. dont le nom est contenu dans SSNAME.

- vérifier l'exactitude du mot de passe contenu dans PSW.

- si la B.D. est fermée au moment de l'appel, la référence B.D. sera retournée à l'utilisateur dans la variable SREF; cette référence sera mémorisée par l'interface et devra etre contenue dans SREF lors de tous les appels ultérieurs, autres qu'une demande d'ouverture.

- ouverture multiple :
  si la B.D. est déjà ouverte SREF sera garni par l'interface avec la référence mémorisée lors de la première demande d'ouverture relative à cette B.D..

- mémoriser le mode d'ouverture des fichiers contenus dans COMOD
  COMOD = 0 : mode automatique;
  COMOD = 1 : mode manuel;

- si COMOD = 0 : mémoriser le mode de protection global contenu dans PROTECT;

  PROTECT = 1 : retrieval;
  PROTECT = 2 : update;
  PROTECT = 3 : protected retrieval;
  PROTECT = 4 : protected update;
  PROTECT = 5 : exclusive retrieval;
  PROTECT = 6 : exclusive update;

- l'ouverture de la B.D. sera réalisée si le mode de protection global contenu dans PROTECT est compatible (cfr définition °) avec le mode de protection global défini au niveau directement supérieur s'il en existe un;

diagnostics d'erreurs :

- " B.D. inconnue " si SSNAME ne contient pas le nom de la B.D. gérée par l'interface;

- " mot de passe incorrect " si PSW ne contient pas le mot de passe de la B.D. gérée par l'interface;

- " mode d'ouverture incorrect " si COMOD est > 1.

- " mode de protection incorrect " si PROTECT est < 1 ou > 6.

- " mode de protection invalide " si PROTECT est incompatible avec le niveau de protection global défini au niveau directement supérieur. (CFR définition de compatibilité)

- Fermeture de la base de données.

entrée :

COP : 12
SREF

sortie :

RETCODE

fonction :

- cloturer les accès à une base de données gérée par l'interface et dont la référence est dans SREF.

- si plusieurs ouvertures de base de données ont été exécutees, celle-ci ne sera physiquement fermée que si le nombre de fermetures (logiques) est égal au nombre d'ouvertures (logiques);
au moment de la fermeture effective (physique), les fichiers encore ouverts seront fermés automatiquement par l'interface.

- si le mode de controle est le mode automatique, tous les fichiers qui ont été ouverts automatiquement le restent sauf s'il s'agit de la dernière fermeture B.D.

diagnostics d'erreurs :

- " B.D. non ouverte " si, au moment de l'appel, la B.D. n'est pas ouverte.

5.1.2. Accès aux fichiers.

- Ouverture des fichiers.

entrée :

COP : 21 / 22
SREF
FILENAME
PROTECT

sortie :

RETCODE
RFIL

fonction :

- initialiser l'accès à un (COP = 22) ou à tous (COP = 21) les fichiers de la base de données dont la référence est dans SREF.

- si COP = 22 : vérifier l'existence du fichier dont la référence est dans FILNAME;
  apres exécution de la primitive, RFIL contiendra la référence attribuee au fichier par l'interface;
  ouverture multiple :
  si le fichier est déjà ouvert RFIL sera garni par l'interface avec la référence mémorisée lors de la première demande d'ouverture relative à ce fichier.

- si COP = 21 : les références attribuées aux fichiers ne seront pas transmises à l'utilisateur;
  c'est pourquoi, il devra user d'un autre moyen pour les connaitre;
  ouverture multiple :
  pour les fichiers déjà ouverts, la référence de ces fichiers sera la meme que celle mémorisée lors de la première demande d'ouverture relative à ces fichiers.

- la protection établie sur le fichier pendant le travail de l'utilisateur sera fonction de la valeur contenue dans PROTECT:

  PROTECT = 1 : retrieval;
  PROTECT = 2 : update;
  PROTECT = 3 : protected retrieval;
  PROTECT = 4 : protected update;
  PROTECT = 5 : exclusive retrieval;
  PROTECT = 6 : exclusive update;

- vérifier que la protection demandée est compatible avec le mode de protection global déterminé lors de l'ouverture de la base de données.

diagnostics d'erreurs :

- "FILENAME invalide " : si FILENAME ne contient pas le nom d'un fichier de la B.D.

- " fichier non disponible " si le fichier dont le nom est dans FILENAME, n'est pas accessible.

- " mode d'ouverture invalide " si l'on tente d'ouvrir un fichier déjà ouvert, avec une protection > à la protection établie lors de la premiere ouverture, ou avec une protection incompatible avec le mode de protection global courant.

- Fermeture des fichiers :

entrée :

COP : 23 / 24
SREF
RFIL

sortie :

RETCODE

fonction :

- cloturer l'accès à un (COP = 24) ou à tous (COP = 23) les fichiers de la base de données dont la référence est dans SREF.

- si COP = 24 : vérifier que RFIL contient la référence à un fichier.

- une fermeture doit toujours avoir été précédée par au moins une une ouverture; cependant COP = 24 sera interprétée comme étant une demande de fermeture de tous les fichiers ouverts.

- une fermeture de fichier n'est effective (physique) que si le nombre de fermetures est égal au nombre d'ouvertures demandées pour ce fichier.

- si la fermeture est effective, le(s) fichier(s) est (sont) devenu(s) inaccessible(s) pour l'utilisateur.

diagnostics d'erreurs :

- " RFIL invalide " si RFIL ne contient pas une référence attribuée à un fichier de la B.D. qui aurait été ouvert avant.

5.1.3. Accès aux articles.

Les primitives suivantes permettent d'accéder à des articles et à leurs valeurs d'items. Elles peuvent toutes etre rassemblées dans une seule catégorie de primitive que nous appelerons :

" l'accès dans une séquence à un sous-ensemble ordonné d'articles "

Les différents types d'accès repris sous cette catégorie se distinguent par la valeur de deux paramètres:

- la définition du sous-ensemble ordonné
- la position de l'article désiré dans le sous-ensemble

Les différents sous-ensembles possibles sont:

- les articles de la base de données
- les articles d'un fichier
- les articles cibles d'un chemin d'accès
- les articles associés à une valeur donnée de clé d'accès, dans un référentiel déterminé, qui peut etre la base de données, un ou plusieurs fichiers, ou un type d'article.

Chacun de ces sous-ensembles peut etre réduit par la sélection à l'intérieur de ceux-ci, des articles d'un type déterminé. Ainsi, nous obtenons 4 autres sous-ensembles possibles:

- les articles d'un type de la base de données
- les articles d'un type d'un fichier
- les articles d'un type cible d'un chemin d'accès
- les articles d'un type associé à une valeur donnée de clé d'accès, dans un référentiel déterminé, qui peut etre la base de données, un ou plusieurs fichiers, ou le type d'article.

Si plus d'un ordre a été déclaré pour ces sous-ensembles, alors un ordre peut etre spécifié.

- naturel: il s'agit de l'ordre utilisé par défaut ou imposé, s'il en existe un.
- chronologique:
  du premier au dernier
  du dernier au premier
- trié:
  dans l'ordre des valeurs d'une clé de tri déterminée;
  cet ordre pouvant etre croissant, décroissant, ...

Comme position de l'article désiré, les possibilités sont les suivantes:

- premier, dernier
- suivant, précédent, tout deux relativement à un article précédemment accédé
- au ième , suivant ou précédant un article précédemment accédé

Ces différents types d'accès sont regroupés en 7 primitives remplissant chacune un certain nombre de fonctions:

- accès séquentiel aux articles de la base de données
- accès par clé aux articles de la base de données
- accès séquentiel aux articles d'un fichier
- accès par clé aux articles d'un fichier
- accès séquentiel aux articles cibles d'un chemin d'accès
- accès par clé aux articles cibles d'un chemin d'accès
- accès par référence à un article

Les 6 premières primitives regroupent toutes les fonctions relatives à un ordre et une position déterminés, tels que nous les avons définis plus haut.

La dernière (accès par référence) est une primitive d'accès par clé particulière, en ce sens qu'elle permet d'accéder un article en donnant une valeur de clé (clé interne) qui l'identifie dans toute la base de données.

De plus, chacune de ces 7 primitives inclut l'accès ou non aux valeurs d'item de l'article que l'on veut accéder.

D'une manière générale, pour les 6 premières primitives, les paramètres suivants détermineront en partie la fonction précise que l'on veut réaliser:

- COREC = 0 ---> on désire accéder aux articles, indépendamment de leur type
- COREC <> 0 ---> on désire accéder aux articles dont le type est celui désigné par COREC
- COGET = 0 ---> on ne désire pas accéder aux valeurs d'items de l'article
- COGET <> 0 ---> on désire accéder aux valeurs d'items de l'article, et ceci par la zone RFIELD

Pour les primitives d'accès séquentiel, la fonction sera également précisée par les paramètres suivants :

- ORDER = 0 ---> on désire accéder aux articles dans leur ordre naturel de création
- ORDER = 1 ---> on désire accéder aux articles selon un ordre trie selon les valeurs d'une clé de tri, dont le code se trouve dans COSIMPLE
- POSIT = 0 ---> on désire accéder au prochain article dans l'ordre déterminé par ORDER et relativement à une posi- donnée par PREF
- POSIT = 1 ---> on désire accéder à l'article précédent dans l'ordre déterminé par ORDER et relativement à une position donnée par PREF
- POSIT = 2 ---> on désire accéder au ième article dans l'ordre déterminé par ORDER et relativement à une position donnée par PREF
  la valeur de i étant contenue dans ZVALUE
- POSIT = 3 ---> on désire accéder au -ième article dans l'ordre déterminé par ORDER et relativement à une position donnée par PREF
  la valeur de i étant contenue dans ZVALUE
- PREF = 0 ---> la position relative est le premier (si POSIT = 0 ou 2) ou le dernier (si POSIT = 1 ou 3) article dans l'ordre déterminé par ORDER

- PREF <> 0 ---> la position relative est l'article dont la référence se trouve dans PREF

- Accès séquentiel aux articles de la base de données:

entrée :

COP : 31
SREF
COREC
COGET
PREF
ORDER
POSIT
ZVALUE
COSIMPLE

sortie :

RETCODE
RREF
COREC
RFIL
RFIELD

fonction :

- fournir la référence de l'article dont le type est éventuellement donné par COREC

- la localisation de l'article est déterminée par les paramètres ORDER, POSIT, PREF, ZVALUE , COSIMPLE

- garnir RFIL de la référence du fichier qui contient cet article.

- si COGET = 1 , placer les valeurs d'items de cet article dans la zone RFIELD

diagnostics d'erreurs :

- " COREC invalide " si COREC est <> 0 et ne contient pas le code d'un type d'article de la B.D.

- " ORDER invalide " si ORDER > 2 ou si ORDER ne correspond pas à un ordre possible pour le type d'accès demandé.

- " COSIMPLE invalide " si COSIMPLE ne contient pas le code d'une clé de tri définie dans la B.D.

- " POSIT invalide " si POSIT > 3.

- " COGET invalide " si COGET > 1

- " fichier non ouvert " si l'on tente d'accéder à un article dans un fichier qui n'à pas été ouvert et que l'on se trouve en mode manuel.

- " article introuvé " si le dernier article auquel on à deja accéde était le dernier dans l'ordre fixé.

- Accès par clé aux articles de la base de données:

entrée :

COP : 32
SREF
COREC
PREF
COSIMPLE
OPERAT
Z-VALUE

sortie :

RETCODE
RREF
RFIL
RFIELD

fonction :

- fournir dans RREF la référence de l'article dont le type est éventuellement spécifie dans COREC , et qui vérifie la condition < COSIMPLE OPERAT Z-VALUE > .

- si PREF = 0, on accéde au premier article vérifiant la condition sinon on accéde au prochain article vérifiant la condition, relativement à un article vérifiant également la condition et dont la référence se trouve dans PREF.

- fournir dans RFIL le code du fichier qui contient cet article.

- si COGET = 1, placer les valeurs d'items de cet article dans la zone RFIELD.

- restriction CODASYL: le type de l'article doit etre obligatoirement spécifié; ainsi, COREC = 0 sera considéré comme une erreur.

diagnostics d'erreurs :

CFR les diagnostics d'erreurs pour l'accès séquentiel aux articles de la B.D. sauf:

- " article introuve " si aucun article satisfaisant la condition

n'a pu etre trouvé.

- Accès séquentiel aux articles d'un fichier:

entrée :

COP : 33
SREF
RFIL
COREC
PREF
COGET
ORDER
COSIMPLE
POSIT
ZVALUE

sortie :

RETCODE
RREF
RFIELD
COREC

fonction :

- fournir dans RREF la référence d'un article, appartenant à un fichier dont la référence se trouve dans RFIL, éventuellement d'un type donné par COREC.

- la localisation de l'article est déterminée par les paramètres ORDER, POSIT, PREF, ZVALUE, COSIMPLE.

- si COREC = 0, le code du type d'article accéde sera transmis dans COREC.

- si COGET = 1, les valeurs d'items de cet article seront places dans la zone RFIELD.

diagnostics d'erreurs :

CFR ceux définis pour l'accès séquentiel aux articles de la base de données avec en plus:

- " RFIL invalide " si RFIL ne contient la référence d'aucun fichier de la B.D.

- Accès par clé aux articles d'un fichier:

entrée :

COP : 34

SREF
RFIL
COREC
PREF
COGET
COSIMPLE
OPERAT
Z-VALUE

sortie :

RETCODE
RREF
RFIELD

fonction :

- fournir dans RREF la référence d'un article, appartenant au fichier dont la référence se trouve dans RFIL, éventuellement d'un type spécifie par COREC, et vérifiant une condition du type: < COSIMPLE OPERAT Z-VALUE >.

- si PREF = 0, on accéde au premier article vérifiant la condition sinon on accéde au prochain article vérifiant la condition, relativement à un article vérifiant également la condition et dont la référence se trouve dans PREF.

- si COGET = 1, placer les valeurs d'items de cet articles dans la zone RFIELD.

- restriction CODASYL: le type de l'article doit etre obligatoirement spécifié; ainsi, COREC = 0 sera considéré comme une erreur.

diagnostics d'erreurs :

CFR les diagnostics d'erreurs pour l'accès par clé aux articles de la B.D.

- Accès séquentiel aux articles cibles d'un chemin.

entrée :

COP : 35
SREF
COSET
COREC
OREF
PREF
ORDER
COSIMPLE
POSIT
ZVALUE
COGET

sortie :

RREF
COREC
RFIL
RFIELD
RETCODE

fonction :

- fournir dans RREF la référence d'un article, cible d'un chemin d'accès dont la référence se trouve dans RFIL, éventuellement d'un type donné par COREC.

- la localisation de l'article est déterminée par les paramètres ORDER, POSIT, PREF, ZVALUE, COSIMPLE, OREF.

- si COREC = 0, le code du type d'article accéde sera transmis dans COREC.

- si COGET = 1, les valeurs d'items de cet article seront places dans la zone RFIELD.

- restriction CODASYL: cette fonction ne vérifiera pas que l'article référencé par PREF, dans le cas ou PREF <> 0, appartient bien à un chemin d'origine OREF; ceci pour un souci d'efficacité.

diagnostics d'erreurs :

CFR ceux définis pour l'accès séquentiel aux articles de la base de données avec en plus:

- " COSET invalide " si COSET ne contient le code d'aucun type de chemin d'accès de la B.D.

- Accès par clé aux articles cibles d'un chemin d'accès:

entrée :

COP : 37
SREF
COSET
COREC
PREF
COGET
COSIMPLE
OPERAT
Z-VALUE

sortie :

RETCODE
RREF
RFIELD
RFIL

fonction :

- fournir dans RREF la référence d'un article, cible d'un chemin d'accès dont la référence se trouve dans RFIL, eventuellement d'un type spécifie par COREC, et vérifiant une condition du type: < COSIMPLE OPERAT Z-VALUE >.

- si PREF = 0, on accéde au premier article vérifiant la condition sinon on accéde au prochain article vérifiant la condition, relativement à un article vérifiant également la condition et dont la référence se trouve dans PREF.

- si COGET = 1, placer les valeurs d'items de cet article dans la zone RFIELD.

- restriction CODASYL: le type de l'article doit etre obligatoirement spécifié; ainsi, COREC = 0 sera considéré comme une erreur.

diagnostics d'erreurs :

CFR les diagnostics d'erreurs pour l'accès par clé aux articles de la B.D. avec en plus:

- " COSET invalide " si COSET ne contient le code d'aucun type de chemin d'accès de la B.D.

- Accès par référence à un article:

entrée :

COP : 38
SREF
COREC
RREF

sortie :

RETCODE
RFIL
RFIELD

fonction :

- si COGET = 1, fournir dans RFIELD les valeurs des items de l'article dont la référence est contenue dans RREF, et dont le type est éven-

tuellement donné par COREC.

- garnir RFIL avec la référence du fichier qui contient cet article.

- si COREC = 0, le code du type d'article que l'on veut accéder, sera transmis dans COREC.

diagnostics d'erreurs :

- " COREC invalide " si COREC <> 0 et ne contient pas le code d'un type d'article de la B.D.

- " RREF invalide " si RREF ne contient pas la référence d'un article dont le type est contenu dans COREC et que COREC <> 0, ou si cet article à été supprimé.

5.2. Les primitives de modifications.

5.2.1. Création d'un article.

entree :

COP : 61
SREF
RFIL
COREC
Z-VALIT
STKREF
SETLST
CURLST

sortie :

RETCODE
RREF

fonction :

- si les contraintes d'intégrité définies dans le schéma peuvent etre vérifiées à l'aide des données d'entrée, enregistrer dans la base de données un article :

  . en accord avec le mode de stockage du type d'article spécifié par COREC, dans le fichier de référence RFIL; STKREF contiendra, si le mode de stockage l'exige (ex. point d'insertion), la référence d'un article;

  . auquel sont associées les valeurs des items contenues dans Z-VALIT;

. ajouté aux chemins d'accès pour lesquels il a été déclaré comme cible ou origine AUTOMATIQUE, en accord avec l'ordre déclaré pour le type de chemin d'accès;
ces chemins sont spécifiés à l'aide des paramètres SETLST et CURLST;

- associe à cet article une DBK en accord avec son mode de stockage dont la valeur est retournée dans RREF;

diagnostics d'erreurs :

- " RFIL invalide " si RFIL ne contient pas la référence d'un fichier de la B.D.
- " COREC invalide " si COREC ne contient pas le code d'un type d'article de la B.D.
- " SETLST invalide " si un des elements de SETLST ne contient pas un code d'un type de chemin d'accès de la B.D.
- " CURLST invalide " si un des elements de CURLST ne contient pas une référence à un article cible ou origine du type de chemin d'accès dont le code se trouve dans l'element de SETLST correspondant.
- " mode d'ouverture invalide " si au moins un fichier affecté par cette primitive n'est pas ouvert en modification (protegee, exclusive ou non).
- " cle double " si certaines valeurs d'items violent l'une ou l'autre contrainte d'identifiant définie dans le schéma.

5.2.2. Suppression d'un article

entree :

- COP : 62
- SREF
- COREC
- RREF
- COMOD

sortie :

- RETCODE

fonction :

- selon le mode de suppression spécifié par COMOD :

  (1) COMOD = 0 : si l'article de reference RREF (de type COREC) n'est origine d'aucun chemin non vide,
  . l'enlever de tous les chemins dont il est cible
  . l'effacer de son fichier;

  (2) COMOD = 1 : . enlever l'article de référence RREF (de type COREC) de tous les chemins dont il est cible,
  . enlever des chemins dont il est origine toutes les cibles optionnelles et appliquer (1) à toutes celles qui sont obligatoires,
  . effacer l'article de son fichier;

  (3) COMOD = 2 : . enlever l'article de référence RREF (de type COREC) de tous les chemins dont il est cible,
  . enlever des chemins dont il est origine toutes les cibles optionnelles qui sont aussi cibles d'autres chemins,
  . pour ces memes chemins, appliquer (3) à toutes les autres cibles (obligatoires ainsi que les optionnelles qui ne sont pas membres d'un autre chemin),
  . l'effacer de son fichier;

  (4) COMOD = 3 : . enlever l'article de référence RREF (de type COREC) de tous les chemins dont il est membre,
  . appliquer (4) à toutes les cibles des chemins dont il est origine,
  . l'effacer de son fichier;

- la référence de l'article supprime peut encore etre utilisee pour realiser l'accès à l'article suivant, et pour cela uniquement.

diagnostics d'erreurs :

- " COREC invalide " si COREC ne contient pas le code d'un type d'article de la B.D.

- " RREF invalide " si RREF ne contient pas la référence d'un article dont le type est specifie dans COREC.

- " COMOD incorrect " si COMOD > 3.

- " suppression invalide " si le mode de suppression ne permet pas de supprimer l'article dans l'etat ou il se trouve.

- " mode d'ouverture invalide " si au moins un fichier affecté par cette primitive n'est pas ouvert en modification (protegee, exclusive ou non).

5.2.3. Modification des valeurs d'item d'un article.

- définitions : item indépendant - item de controle

par la suite, nous entendrons par "item de controle", tout item associé à

. un ordre

et/ou

. une clé d'accès

et/ou

. un identifiant

par opposition, tout item n'ayant pas cette caratéristique sera appelé "item indépendant";

un "groupe de controle" est un ensemble d'item de controle et correspondra selon les cas à un ordre et/ou une clé d'accès et/ou un identifiant.

- modification de valeur d'item independant:

entree :

- COP : 71
- SREF
- COREC
- RREF
- Z-VALIT

sortie :

- RETCODE

fonction :

- remplacer les valeurs des items indépendants de l'article dont référence est dans RREF et le type dans COREC, par les valeurs des items indépendants de l'article, contenues dans Z-VALIT.

- les items modifiés sont tous les items independants associes à l'article.

- rem : la zone Z-VALIT à la meme structure que celle qui à servi à créer l'article, et donc contient également les sous-zones destinées à recevoir les valeurs des items non indépendants (items de controle); cependant, aucune vérification ne sera faite pour savoir si le contenu de ces sous-zones à été modifié vu que cette primitive les ignore tout simplement.

diagnostics d'erreurs :

- " COREC invalide " si COREC ne contient pas le code d'un type d'article de la B.D.

- " RREF invalide " si RREF ne contient pas une référence à un article dont le type se trouve dans COREC.

- " mode d'ouverture invalide " si au moins un fichier affecté par cette primitive n'est pas ouvert en modification (protegee, exclusive ou non).

- modification d'un groupe de controle:

entree :

- COP : 72
- SREF
- COREC
- RREF
- COSIMPLE
- Z-VALIT

sortie :

- RETCODE

fonction :

- remplacer la valeur des items associés au groupe de controle dont le code se trouve dans COSIMPLE;
  ce groupe doit etre associé à un article dont le type se trouve dans COREC et la référence dans RREF.

- les valeurs des items à modifier doivent se trouver dans les sous-zones de Z-VALIT correspondantes.

diagnostics d'erreurs :

- " COREC invalide " si COREC ne contient pas le code d'un type d'article de la B.D.

- " RREF invalide " si RREF ne contient pas la référence d'un article dont le type est dans COREC.

- " COSIMPLE invalide " si COSIMPLE ne contient pas le code d'un groupe de controle associe à l'article dont le type est dans COREC.

- " mode d'ouverture invalide " si au moins un fichier affecté par cette primitive n'est pas ouvert en modification (protegee, exclusive ou non).

- " cle double " si certaines valeurs d'items violent l'une ou l'autre contrainte d'identifiant définie dans le schéma.

5.2.4. Insertion d'un article dans un chemin.

entree :

- COP : 81
- SREF
- COSET
- RREF
- SRREF

sortie :

- RETCODE

fonction :

- insérer l'article dont la référence est dans RREF, dans le chemin d'accès dont le type se trouve dans COSET, en accord avec l'ordre déclaré pour ces chemins.

- SRREF indiquera, si l'ordre déclaré pour le chemin l'exige, un point d'insertion relatif.

diagnostics d'erreurs :

- " COSET invalide " si COSET ne contient pas le type d'un chemin d'accès de la B.D. pour lequel une insertion manuelle pour au moins une de ses cibles est autorisee.

- " SRREF invalide " si SRREF ne contient pas la référence d'un article pouvant faire l'objet d'un point d'insertion et que l'ordre defini pour le chemin d'accès necessite la definition d'un tel point.

- " RREF invalide " si RREF ne contient pas la référence d'un article pouvant etre cible du chemin d'accès dont le type est designe par COSET.

- " mode d'ouverture invalide " si au moins un fichier affecté par cette primitive n'est pas ouvert en modification (protegee, exclusive ou non).

5.2.5. Retrait d'un article d'un chemin d'accès.

entree :

- COP : 82
- SREF
- COSET
- OREF
- RREF

sortie :

- RETCODE

fonction :

- retirer un article dont la référence se trouve dans RREF, du chemin d'accès dont la référence de l'origine se trouve dans OREF, et le type dans COSET.

diagnostics d'erreurs :

- " COSET invalide " si COSET ne contient pas le code d'un type de chemin d'accès de la B.D. pour lequel le retrait d'au moins un de ses articles cibles est autorise.

- " OREF invalide " si OREF ne contient pas la référence d'un article origine d'un chemin d'accès dont le type se trouve dans COSET.

- " RREF invalide " si RREF ne contient pas la référence d'un article cible d'un chemin d'accès dont le type se trouve dans COSET, et la référence de l'origine dans OREF, ou que cet article cible a été declare comme devant appartenir 'obligatoirement' au chemin d'accès.

- " mode d'ouverture invalide " si au moins un fichier affecté par cette primitive n'est pas ouvert en modification (protegee, exclusive ou non).

5.2.6. Changement de chemin d'accès.

entree :

- COP : 83
- SREF

- COSET
- OREF
- SRREF
- RREF

sortie :

- RETCODE

fonction :

- retirer un article cible d'un chemin d'accès et l'insérer dans un autre chemin du meme type.

- la référence de l'article se trouve dans RREF

- le code du type de chemin dans lequel doit se faire le transfert, se trouve dans COSET; la référence de l'article origine du chemin duquel on doit retirer l'article se trouve dans OREF; si l'ordre d'insertion defini pour le type de chemin impose la determination d'un point d'insertion, celui-ce sera contenu dans SRREF.

- " mode d'ouverture invalide " si au moins un fichier affecté par cette primitive n'est pas ouvert en modification (protegee, exclusive ou non).

- restriction CODASYL : fonction non disponible.

5.3. Les primitives de controle.

Ces primitives ont pour objet de controler et commander les mécanismes chargés de la gestion de la concurrence et de la sécurité. Ces fonctions n'ont pu etre implémentées pour les interfaces DBMS-20. Ainsi, toute tentative d'accès à ces primitives se soldera par une valeur de RETCODE signifiant leur non implémentation.

Des exemples de ces fonctions de controle seraient :

- demande de controle sur un article fonction :

  - vérifier l'existence d'un article
  - fournir le type d'un article
  - bloquer un article

- demande de libération d'un article fonction :

6. TYPE ET FORMAT DES PARAMETRES D'APPEL DE L'INTERFACE

```
01 Z-CODES.
     02 COP.
        03 FCOP              PIC 9   DISPLAY-9.
        03 SCOP              PIC 9   DISPLAY-9.
     02 SREF                 PIC 99  DISPLAY-9.
     02 RFIL                 PIC 99  DISPLAY-9.
     02 RETCODE.
        03 FNCODE            PIC 99  DISPLAY-9.
        03 ERRCODE           PIC 99  DISPLAY-9.
     02 PROTECT              PIC 9   DISPLAY-9.
     02 COGET                PIC 9   DISPLAY-9.
     02 CONTRL               PIC 9   DISPLAY-9.
     02 COREC                PIC 99  DISPLAY-9.
     02 RREF                 PIC S9(10) COMP.
     02 PREF                 PIC S9(10) COMP.
     02 COSIMPLE             PIC 99  DISPLAY-9.
     02 OPERAT               PIC 9   DISPLAY-9.
     02 COMOD                PIC 9   DISPLAY-9.
     02 COSET                PIC 99  DISPLAY-9.
     02 OREF                 PIC S9(10) COMP.
     02 TYP                  PIC X   DISPLAY-9.

01 Z-IDENT.
     02 NAME                 PIC X(30)
     02 PSW                  PIC X(30).
     02 FILLER               PIC X(int1).

01 ZVALUES REDEFINES Z-IDENT.
     02 ZVALUE               PIC X(int2).

01 Z-ITEM.
     02 ITEMLST.
        03 ITEM              PIC 9(4) OCCURS int3.

01 Z-RESP.
     02 RFIELD               PIC X(int2).

01 Z-SETS.
     02 STKREF               PIC S9(10) COMP.
     02 SETLST.
        03 SETL              PIC 99  DISPLAY-9 OCCURS int4.
     02 CURLST.
        03 CURNT             PIC S9(10) COMP   OCCURS int4.
     66 SRREF RENAMES STKREF.
```

int1 : longueur du plus grand type d'article de la B.D. (BDLGMX) - 60
int2 : BDLGMX
int3 : nombre d'item dans la base de données
int4 : nombre de type de chemins dans la base de données

7. CODES D'ERREURS.

Nous énumérons ici les valeurs de RETCODE qui seront transmises à l'utilisateur lors de la détection d'erreurs par l'interface.

22 : clé double
23 : COGET non autorisé (sur un article SYSTEM)
26 : pas d'article trouvé
30 : fichier non disponible
66 : COREC et RREF sont incompatibles
67 : COSET et OREF ou STKREF sont incompatibles
68 : RFIL et COREC sont incompatibles
69 : COREC et COSET sont incompatibles
70 : COSET incorrect
71 : POSIT incorrect
72 : COGET incorrect
75 : OPERAT incorrect
77 : mode d'ouverture incompatible (fichier ou B.D.)
78 : PREF incorrect
79 : COMOD incorrect
80 : fichier non ouvert
88 : COSIMPLE incorrect
89 : fonction non disponible
90 : B.D. non accessible
91 : RFIL incorrect
92 : PROTECT incorrect
93 : PSW incorrect
94 : FILENAME incorrect
95 : B.D. non ouverte
96 : COREC incorrect
97 : SREF incorrect
98 : BDNAME incorrect
99 : COP incorrect

8. CODES OPERATIONS.

Nous rappelons ci-dessous les codes opérations attribués à chacune des fonctions susceptibles d'etre réalisées par les interfaces.

Une "*" à devant un code opération signalera celles qui ne sont pas disponibles dans les interfaces DBMS-20.

11 : ouverture B.D.
12 : fermeture B.D.
21 : ouverture de tous les fichiers
22 : ouverture d'un fichier
23 : fermeture de tous les fichiers
24 : fermeture d'un fichier
31 : accès séquentiel aux articles (d'un type) de la base de données
32 : accès par clé aux articles d'un type dans la base de données
33 : accès séquentiel aux articles (d'un type) dans un fichier
34 : accès par clé aux articles d'un type dans un fichier
35 : accès séquentiel aux articles (d'un type) cibles d'un chemin
37 : accès par clé aux articles d'un type cibles d'un chemin
38 : accès par référence à un article
* 51 : controle B.D.
* 52 : controle fichier
* 53 : controle article
61 : création d'un article
62 : suppression d'un article
71 : modification des valeurs des items indépendants d'un article
72 : modification des valeurs des items de controle d'un article
81 : insertion d'un article dans un chemin
82 : retrait d'un article d'un chemin

* 83 : transfert d'un article dans un autre chemin

MANUEL D'UTILISATION
DE L'ANALYSEUR

août 1982

Introduction.

Cette annexe est consacrée à l'utilisation de l'analyseur de description de schéma pour B.D. gérées par DBMS-20 (cfr chapitre 7).

Le plan de notre exposé sera le suivant :

. utilisation pratique de l'analyseur

. affectation des codes externes

. restrictions imposées sur le DDL

. adaptation de l'analyseur à des modifications du DDL

1. Utilisation pratique de l'analyseur.

1.1. Description du schéma (DDL)

Avant toute chose, il faut disposer d'une description correcte du schéma pour lequel on veut générer un interface. Cette description, en plus d'etre correct, devra respecter les restrictions énoncées plus bas et contiendra des éventuelles affectations de codes externes (cfr 2.).

Pour la version COBOL implantée sur le DEC-20, l'affectation des fichiers pour un programme se fait par leur nom physique et non logique. L'analyseur qui doit tourner sur cette version doit donc tenir compte de cette restriction. Ainsi la description du schéma, pour pouvoir etre analysée, doit etre contenue dans un fichier appelé FDDL. Les descriptions des différents fichiers utilisés sont repris dans le manuel d'utilisation de la méta-BD (annexe).

1.2. La B.D. schéma

Rappelons que l'analyseur est chargé de garnir la B.D. schéma (cfr chapitre 4) en fonction d'une description de schéma. Certaines dispositions sont à prendre selon que cette B.D. schéma existe déjà ou n'existe pas encore.

a) si la B.D. schéma n'existe pas :

Il faut l'initialiser. Dans le cas du DEC-20, cela se fait par l'initialisation des fichiers indexés qui la consitituent. Ceci ce fait au moyen de l'utilitaire ISAM. L'initialisation doit se faire en respectant les déclarations des fichiers COBOL (cfr manuel d'utilisation de la méta-BD). Nous donnons ici un tableau (fig 1) reprenant les options prises pour les différents fichiers, lors de l'exécution de l'utilitaire ISAM.

	méta-objets	méta-chemins	particularités	sous-schémas
àISAM				
*file-name,file-name=NULL/B	MOBJET	MCHEMI	MPARTI	MFILES
Mode of input file:	A	A	A	A
Mode of output file:	A	A	A	A
Maximum record size:	100	13	126	85
Key descriptor:	X1.7	X1.10	X1.6	X7.30
Records per input block:	0	0	0	0
Total records per Data block (Recommended: --)	64	32	109	128
Empty records per Data block:	0	0	0	0
Total Entries per Index block (Recommended: --)	--	--	--	--
Empty entries per index block:	2	2	2	2
Percentage of Data file to leave empty:	0	0	0	0
Percentage of Index file to leave empty:	0	0	0	0
Maximum number of records file can become:	?	?	?	?
...				
^C				

b) si la B.D. schéma existe déjà :

En principe il ne faut rien faire. Cependant, il est conseillé de travailler sur une copie de la B.D. schéma car aucun moyen de restauration n'a été réalisé au cas ou un accident arriverait durant l'exécution de l'analyseur.

1.3. Exécution de l'analyseur.

Le programme à exécuter s'appelle ANALYS.CBL . Lors de l'exécution, pour chaque clause principale analysée (SCHEMA, AREA, RECORD, SET, SUB-SCHEMA), un message sera affiché. Il aura la forme : DDL <clause-name> . Ces messages sont prévus pour aider l'utilisateur à se localiser dans la description, en cas d'erreur.

Pour chaque déclaration de sous-schéma, il sera demandé le nom du programme interface qui lui sera associé. Ce nom est composé de 5 lettres au maximum. C'est ce meme nom qui figurera dans la clause PROGRAM-ID de l'interface.

Si l'exécution est réussie, le message "NORMAL EXIT" sera affiché. Dans le cas contraire, l'exécution aura été interrompue soit, par une erreur détectée par l'analyseur, soit par une erreur d'exécution de cause inconnue par l'analyseur.

1.4. Erreurs détectées par l'analyseur.

. FDDL-line
"CLAUSE NOT ADMITTED BY ANALYZER"

Cette erreur est détectée lorsque l'analyseur a lu une clause qui lui est inconnue. Il s'agira en principe des clauses qui font l'objet d'une restriction (cfr 3.).

. "ERROR IN EXTERNAL CODE"

Détecté lors d'une erreur d'affectation de code externe (cfr 2.) .

. "ERROR: END-OF-DDL DETECTED BEFORE "END-SCHEMA" "

Détecté lorsque l'analyseur trouve la fin du fichier contenant la description avant d'avoir lu "END-SCHEMA". Cette erreur peut se produire lorsque l'analyseur est occupé à traiter une clause, et qu'il ne trouve pas un mot "obligatoire" dans cette clause, par exemple un mot terminal tel que un "." .

Les messages qui suivent concernent des erreurs d'écritures sur les fichiers de la B.D. schéma. En principe, elles ne peuvent se déclencher que s'il existe dans la description, des objets portant le meme identificateur.

. "ERROR IN WRITING OBJECT - INVALID KEY"

. "ERROR IN WRITING RELATION - INVALID KEY"

. "ERROR IN WRITING PARTICULARITY - INVALID KEY"

. "ERROR IN WRITING ON FFILE - INVALID KEY"

2. Affectation des codes externes.

Les codes externes sont des codes attribués à certains objets (types d'articles, items, types de chemins, clés d'accès) en vue de les désigner lors de certains appels aux interfaces (cfr manuel d'utilisation des interfaces et chapitre 7). Nous donnons ici les règles qui permettent aux utilisateurs d'attribuer ces codes dans la description du schéma.

Ces règles sont les suivantes :

a) Seuls peuvent se voir attribuer des codes externes les types d'articles, les items, les types de chemins et les clés d'accès.

b) Un code externe est identifiant pour les objets d'un meme type.

c) Une attribution aura le format suivant :

(#integer#) ou 'integer' est un entier compris entre 1 et 99

Rappel : Un code externe attribué à un item est un code relatif au type d'article auquel il est associé; le code complet d'un item est constitué du code externe du type d'articles contenant l'item et du code relatif de l'item.

d) L'attribution d'un code se rapportera toujours au dernier identificateur d'objet lu par l'analyseur; ainsi l'attribution d'un code suivra l'identification de l'objet auquel il est associé mais précédera l'identification d'un autre objet.

Exemples d'attributions :

. pour un type d'articles :

```
RECORD NAME IS produit  (#5#)
LOCATION MODE IS ...
```

. pour un item :

```
02 numero-produit  PIC 9(7) USAGE IS DISPLAY-7.  (#3#)
```

. pour un type de chemins :

SET NAME IS com-cli (#12#)

. pour une clé d'accès (simple) :

LOCATION MODE IS CALC USING nom-cli, pre-cli, adr-cli (#7#)

ASCENDING KEY IS num-com-cc, qte-cc (#21#)

Il est toutefois à noter que ces attributions ne sont nullement obligatoires; elles peuvent etre associées à l'ensemble des objets de la description susceptibles d'en recevoir, à une partie de ceux-ci ou à aucun de ceux-ci.

Dans le cas ou aucun code n'a été attribué pour un certain type d'objet, ils seront attribués séquentiellement par l'analyseur à partir de 1.

Si certains codes ont été attribués aux objets d'un type, l'analyseur attribuera aux objets n'en ayant pas recu un code différent de ceux déjà attribués et le plus petit possible.

3. Restrictions imposées sur le DDL.

Faute de temps, nous nous sommes vus obligés de restreindre quelque peu le DDL accepté par l'analyseur. Cependant, il nous semble que dans la plupart des cas, ces restrictions n'affecteront que peu les désirs des utilisateurs. D'autre part, il est toujours possible, pour les utilisateurs insatisfaits, d'y remédier en suivant les directives de modification de l'analyseur énoncées au point 4.

Ces restrictions sont les suivantes :

SCHEMA DATA ENTRY :

Seul le format 1 est autorisé, et dans ce format, la clause USAGE n'est pas analysée et le 'picture-string' ne peut contenir que des 9, A, X, V, (,) .

SCHEMA MEMBER ENTRY :

Pour la SET OCCURENCE SELECTION, seule l'option THRU CURRENT OF SET est admise.

La clause MEMBER doit se terminer par un '.' .

SUB-SCHEMA ENTRY :

Dans la RECORD SECTION, le format 1 est admis uniquement s'il ne contient que des clauses '01 record-name'. Toutes les possibilités concernant les items ne sont pas autorisées.

4. Adaptation de l'analyseur à des modifications du DDL.

Il est indispensable que l'analyseur puisse etre adapté suite à des modifications du DDL. Nous donnons ici quelques directives destinées à aider l'utilisateur lors de ces adaptations. Nous supposerons que celui-ci a déjà pris connaissance de ce qui est dit au chapitre 7 à propos du principe de fonctionnement de l'analyseur. Nous ne rappelerons que ces deux points :

- le DDL est composé de clauses et sous-clauses
- l'analyseur est composé de procédures (automates) chacune d'elles destinée à analyser une clause ou sous-clause;

Les différentes procédures ou paragraphes chargés d'analyser les clauses portent en général un nom rapproché de celui de la clause. Par exemple, pour la SCHEMA ENTRY, nous trouvons une procédure TR-SCHEMA, etc.

La partie "analyse" du programme est décomposée en plusieurs niveaux. Ces niveaux correspondent aux niveaux d'imbrications des clauses et sous-clauses du DDL. Ainsi, au niveau 1 on trouvera des procédures relatives aux clauses SCHEMA, AREA, RECORD, SET et SUB-SCHEMA; au niveau 2 des paragraphes relatifs aux sous-clauses telles que LOCATION MODE DIRECT, WITHIN, DATA, ... et ainsi de suite jusqu'au niveau le plus bas.

De cette manière, lorsqu'il faut localiser le paragraphe relatif à une clause ou sous-clause déterminée, il suffit de regarder le niveau d'imbrication de la clause pour s'y retrouver.

Chaque paragraphe a en général la forme suivante (*):

. initialisation :
  . initialisations de certaines variables
  . incrémentations de compteurs
  . positionnement du lecteur sur la première 'information'

. traitement de(s) information(s)

. terminaison : positionnement du lecteur sur le prochain 'sélecteur'

Nous avons vu (chapitre 7) que les clauses étaient composées de mots qui pouvaient etre classés en deux catégories: les terminaux et les non terminaux. Une clause est identifiée par un groupe de mots que l'on appelle 'sélecteur'.

L'analyseur comprend une table contenant les 'sélecteurs' et les mots terminaux facultatifs associés aux différentes clauses. Cette table est constituée par un ensemble de conditions (cfr rubrique 88) portant sur le mot qui vient d'etre lu MOT (pour les terminaux facultatifs) et sur le prochain mot qui sera lu NEXT-MOT (pour les sélecteurs). Ces conditions portent des noms 'B-DUMMY-clause-name' pour les terminaux facultatifs et 'B-SELECT-clause-name' pour les 'sélecteurs'.

Directives :

1. Modification d'une clause.

Une modification de clause consistera en général en une modification, suppression ou insertion de mots qui la composent. S'il s'agit de mots terminaux facultatifs ou de 'sélecteurs', la modification se fera au niveau des conditions portant sur les variables MOT et NEXT-MOT.

2. Suppression d'une clause.

Dans ce cas, il faut d'abord localiser, dans le programme, la procédure qui traite la clause en question. Ceci se fait à l'aide du nom de la clause. Ensuite il suffit de supprimer cette procédure, tous les appels à cette procédure et adapter les procédures relatives aux clauses pouvant précéder la clause à supprimer. Cette adaptation est à réaliser sur la 'terminaison' de ces procédures.

3. Ajout d'une clause.

Adapter les 'terminaisons' des procédures relatives aux clauses pouvant précéder la clause. Ajouter une procédure traitant la clause en suivant la forme (*).

MANUEL D' UTILISATION DE LA META-BASE DE DONNEES "METABD"

Juillet 1982

TABLE DES MATIERES

1. INTRODUCTION

Ce manuel suppose que le lecteur a pris connaissance du quatrième chapitre du mémoire et on ne peut espérer une complète compréhension de ce manuel que si l'on a lu ce chapitre.

Le méta-base de données est une base de données d'accès et elle n'admet pas les primitives de modification ou de contrôl.

Les accès sont fournis par une interface d'accès. Dans le deuxième chapitre de ce manuel sera exposée la manière dont il faut utiliser cette interface.

Le SGBD ne prenant pas en charge lui-même la mise-à-jour de la base de données, l'utilisateur doit se charger lui-même de garnir les structures de données. Dans le troisième et quatrième chapitre sera exposé l'implémentation des structures de données et il sera données des directives à l'utilisateur pour stocker de l'information dans la méta-base de données.

2. INTERFACE D' ACCES

2.1. LE LANGAGE

2.1.1. INTRODUCTION

Le langage utilisé pour accéder à la méta-base de données est un sous-ensemble du langage utilisé par les interfaces d'accès. Ce langage étant bien documenté dans "Le manuel d'utilisation des interfaces", nous nous contentons ici de préciser quelles sont les fonctions retenues et de rappeler les paramètres d'appel. La description des paramètres est aussi donnée dans ce manuel d'utilisation et par conséquent elle n'est pas reprise ici.

2.1.2. LES FONCTIONS

a) ouverture de la base de données

entrée :

COP : 11
SSNAME
PSW
COMOD
PROTECT

sortie :

SREF
RETCODE

b) fermeture de la base de données

entrée :

COP : 12
SREF

sortie :

RETCODE

c) ouverture d'un fichier (sous-schéma)

entrée :

COP : 21
SREF
FILENAME
PROTECT

sortie :

RETCODE
RFIL

d) fermeture d'un fichier

entrée

COP : 24
SREF
RFIL

sortie :

RETCODE

e) accès aux articles d'un type dans un fichier

entrée :

COP : 33
SREF
RFIL
COREC
COGET : 1
PREF

sortie :

RETCODE
RREF
COREC
RFIL
RFIELD

f) accès par clé aux articles d'un type d'un fichier

entrée :

COP : 34
SREF
RFIL
COREC
COGET : 1
PREF
COSIMPLE
OPERAT : 1
Z-VALUE

sortie :

RETCODE
RREF
RFIL
RFIELD

g) accès sequentiel aux articles cibles d'un chemin

entrée :

COP : 35
SREF
RFIL
COSET
COREC
OREF
PREF
COGET : 1

sortie :

RREF
COREC
RFIELD
RETCODE

2.2. LES CODES DES OBJETS

2.2.1. CODES DES TYPES DE META-ARTICLES

Les types de mêta-articles suivis de leurs codes:

BD	1
FICHIER	2
TARTICLE	3
TCHEMIN	4
ITEM	5
ORIGINE	6
CIBLE	7
GLOBAL	8
SIMPLE	9
COMPOSANT	10

2.2.2. CODES DES TYPES DE META-CHEMINS

Les types de méta-chemins suivis de leurs codes:

BDSI (bd --> simple)	1
FIBD (fichier --> bd)	2
FITA (fichier --> tarticle)	3
FIGL (fichier --> global)	4
FISI (fichier --> simple)	5
TAFI (tarticle --> fichier)	6
TAOR (tarticle --> origine)	7
TACI (tarticle --> cible)	8
TAIT (tarticle --> item)	9
TASI (tarticle --> simple)	10
TCOR (tchemin --> origine)	11
TCCI (tchemin --> cible)	12
TCGL (tchemin --> global)	13
TCSI (tchemin --> simple)	14
TCCO (tchemin --> composant)	15
ITCO (item --> composant)	16
SICO (simple --> composant)	17
ITIT (item --> item)	18
GLSI (global --> simple)	19

2.2.3. CODES DES TYPES DE META-CLES D' ACCES

Les types de mèta-clès suivis de leurs codes:

BDCODE	1
BDIDEN	2
FICODE	3
FIIDEN	4
TACODE	5
TAIDEN	6
TCCODE	7
TCIDEN	8
ITCODE	9
ITIDEN	10
GLCODE	11

Remarque: toutes les clès sont identifiantes.

2.3. LES ERREURS

2.3.1. LES ERREURS SELON LE MODELE D' ACCES

26 pas d'article trouvé
72 COGET incorrect
75 OPERAT incorrect
78 RREF incorrect
88 COKEY incorrect
91 RFIL incorrect
94 FILENAME incorrect
95 BD non-ouverte
96 COREC incorrect
97 SREF incorrect
99 COP incorrect

2.3.2. LES ERREURS SPECIFIQUES A L' INTERFACE

9910 le nombre de méta-fichiers qui peuvent être accédés en même temps est dépassé
9911 les capacités en place mémoire de l'interface sont épuisées; on ne peut plus recevoir le nouveau méta-fichier
9912 il y a trop d'ouvertures multiples sur un même méta-fichier; la nouvelle demande n'est pas acceptée
9913 il y a trop d'ouvertures multiples sur la base de données; la nouvelle demande n'est pas acceptée
9922 une erreur a été détectée sur les fichiers de la base de données permanente
9923 tentative de fermer la base de données alors qu'elle n'est pas ouverte
9924 tentative de fermer un méta-fichier alors qu'il n'est pas ouvert

2.4. UTILISATION AVEC TOPS-20

2.4.1. INTRODUCTION

Cette partie du manuel doit aider l'utilisateur à interoger la méta-base de données à partir d'un programme utilisateur sur TOPS-20.

L'interface est un sous-programme COBOL qui doit être appelé à partir d'un programme utilisateur COBOL. L'appel se fait donc par un CALL ... USING qui a la forme suivante:

```
CALL METABD USING z-codes, z-ident, z-value, z-resp.
```

Les paramètres d'appel de la méta-base de données ont le format suivant :

```
01  Z-CODES.
    02 COP          PIC 99.
    02 SREF         PIC X.
    02 COREC        PIC 99.
    02 RETCODE      PIC 9999.
    02 PROTECT      PIC 9.
    02 COGET        PIC 9.
    02 CONTRL       PIC 9.
    02 RFIL         PIC 99.
    02 RREF         PIC 9(10).
    02 PREF         PIC 9(10).
    02 COSIMPLE     PIC 99.
    02 OPERAT       PIC 9.
    02 COMOD        PIC 9.
    02 COSET        PIC 99.
    02 OREF         PIC 9(10).
    02 TYP          PIC X.
01  Z-IDENT.
    02 SSNAME       PIC X(30).
    02 PSW          PIC X(30).
01  Z-VALUE         PIC X(32).
01  Z-RESP.
    02 RFIELD       PIC X(90).   zone méta-article
    02 PFIELD.                   zone particularités
     03 FILLER      PIC 9(15).
     03 FILLER      PIC 9(15).
     03 FILLER      PIC X(90).
```

2.4.2. LES FICHIERS

L'interface d'accès a en entrée quatre fichiers différents. Ces fichiers contiennent la base de données permanente. Il faut que ces quatre fichiers existent. Ces fichiers ne peuvent pas être créés et maintenus par l'interface mais par d'autres programmes.

Ces fichiers sont décrits dans le chapitre quatre de ce manuel. Il est rappelé cependant ici qu'il s'agit des fichiers suivants:

MOBJET.IDX
MCHEMI.IDX
MPARTI.IDX
MFILES.IDX

3. LES TYPES DE META-ARTICLES ET LEURS ITEMS

Pour chaque type de méta-article nous donnons tous items ainsi qu'une description de leur représentation. La sémantique des méta-items peut être trouvée au chapitre quatre du mémoire.

RFIELD contiendra un méta-article du type demandé si la recherche dans la base de données a été un succès. Dans la suite nous donnons donc un description de RFIELD selon le type de méta-article demandé.

```
le méta-article BD :
     02 BDPNTR      PIC 9999.
     02 BDCODE      PIC 99.
     02 BDIDEN      PIC X(30).
     02 BDPASW      PIC X(30).
     02 BDINTE      PIC X(6).
     02 BDOP01      PIC 9.
     02 BDOP02      PIC 9.
     02 BDOP03      PIC 9.
     02 BDOP04      PIC 9.
     02 BDNBFI      PIC 99.
     02 BDNBTC      PIC 99.
     02 BDNBTA      PIC 99.
     02 BDNBTA      PIC 99.
     02 BDLGMX      PIC 999.
     02 FILLER      PIC X(7).

le méta-article FICHIER :
     02 FIPNTR      PIC 9999.
     02 FICODE      PIC 99.
     02 FIIDEN      PIC X(30).
     02 FIOP01      PIC 9.
     02 FIOP02      PIC 9.
     02 FIOP03      PIC 9.
     02 FIOP04      PIC 9.
     02 FIOP05      PIC 9.
     02 FIOP06      PIC 9.
     02 FIOP07      PIC 9.
     02 FIOP08      PIC 9.
     02 FINBTA      PIC 99.
     02 FILLER      PIC X(64).

le méta-article TARTICLE :
     02 TAPNTR      PIC 9999.
     02 TACODE      PIC 99.
     02 TAIDEN      PIC X(30).
     02 TAOP01      PIC 9.
     02 TAOP02      PIC 9.
     02 TAOP03      PIC 9.
     02 TAOP04      PIC 9.
     02 TAOP05      PIC 9.
     02 TALONG      PIC 999.
     02 TANBFI      PIC 99.
     02 TANBOR      PIC 99.
```

```
     02 TANBCI      PIC 99.
     02 TANBIT      PIC 99.
     02 TANBSI      PIC 99.
     02 FILLER      PIC X(36).

type de méta-article TCHEMIN :
     02 TCPNTR      PIC 9999.
     02 TCCODE      PIC 99.
     02 TCIDEN      PIC X(30).
     02 TCCONN      PIC 9.
     02 TCINVE      PIC 999.
     02 TCNBOR      PIC 99.
     02 TCNBCI      PIC 99.
     02 TCNBGL      PIC 99.
     02 TCNBSI      PIC 99.
     02 TCNBCO      PIC 99.
     02 FILLER      PIC X(40).

type de méta-article ITEM :
     02 ITCODE.
     03 ITCOTA      PIC 99.
     03 ITCOIT      PIC 99.
     02 ITIDEN      PIC X(30).
     02 ITOBFA      PIC 9.
     02 ITRPMX      PIC 999.
     02 ITCMPT      PIC 99.
     02 ITOP01      PIC 9.
     02 ITOP02      PIC 9.
     02 ITNOLV      PIC 99.
     02 ITSTRU      PIC 9.
     02 ITUNIT      PIC 9.
     02 ITLONG      PIC 999.
     02 ITDECI      PIC 99.
     02 ITCOAS      PIC 99.
     02 ITNBIT      PIC 99.
     02 ITNBCO      PIC 99.
     02 FILLER      PIC X(29).

type de méta-article ORIGINE :
     02 ORPNTR      PIC 9999.
     02 ORMINS      PIC 9.
     02 ORMRET      PIC 9.
     02 OROP01      PIC 9.
     02 OROP02      PIC 9.
     02 OROP03      PIC 9.
     02 OROP04      PIC 9.
     02 ORCOTC      PIC 99.
     02 ORCOTA      PIC 99.
     02 FILLER      PIC X(76).

type de méta-article CIBLE :
     02 CIPNTR      PIC 9999.
     02 CIMINS      PIC 9.
     02 CIMRET      PIC 9.
     02 CIOP01      PIC 9.
     02 CIOP02      PIC 9.
     02 CIOP03      PIC 9.
```

```
     02 CIOP04      PIC 9.
     02 CICOTC      PIC 99.
     02 CICOTA      PIC 99.
     02 FILLER      PIC X(76).

type de méta-article GLOBAL :
     02 GLPNTR      PIC 9999.
     02 GLCODE      PIC 99.
     02 GLCLKE      PIC 9.
     02 GLCLOR      PIC 9.
     02 GLCLID      PIC 9.
     02 GLTYRF      PIC 9.
     02 GLREFE      PIC 999.
     02 GLORDE      PIC 9.
     02 GLDBLE      PIC 9.
     02 FILLER      PIC X(75).

type de méta-article SIMPLE :
     02 SIPNTR      PIC 9999.
     02 SICODE      PIC 99.
     02 SICLKE      PIC 9.
     02 SICLOR      PIC 9.
     02 SICLID      PIC 9.
     02 SITYRF      PIC 9.
     02 SIREFE      PIC 99.
     02 SIORDE      PIC 9.
     02 SIDBLE      PIC 9.
     02 SICOTA      PIC 99.
     02 FILLER      PIC X(72).

type de méta-article COMPOSANT :
     02 COPNTR      PIC 9999.
     02 COTYPE      PIC 9.
     02 COSENS      PIC 9.
     02 COCOSI      PIC 99.
     02 COITTC      PIC 9999.
     02 FILLER      PIC X(78).
```

Si xxPNTR est garni, un enregistrement de particularités est attaché au méta-article. Ces particularités sont fournies à l'utilisateur dans PFIELD, qui a la structure suivante:

```
     02 PANUM1      PIC 9(6).
     02 PANUM2      PIC 9(6).
     02 PANUM3      PIC 9(6).
     02 PANUM4      PIC 9(6).
     02 PANUM5      PIC 9(6).
     02 PAALP1      PIC X(30).
     02 PAALP2      PIC X(30).
     02 PAALP3      PIC X(30).
```

4. CREATION

4.1. INTRODUCTION

Le système de gestion de la méta-base de données est un système "read-only" et ne s'occupe donc pas de créer et de maintenir la méta-base de données. Si l'utilisateur veut donc mettre à jour la méta-base, il doit le faire à l'aide de programmes spécialisés. Si les programmes existants de mise à jour ne lui conviennent pas il peut être amené à en écrire un soi-même. Cette partie doit fournir toute l'information nécessaire concernant les fichiers pour écrire un tel programme.

Pour chacun des fichiers nous donnons

- description en FILE-CONTROL
- description en FILE SECTION
- informations supplémentaires

4.2. FICHIER DES META-FICHIERS

Ce fichier contient un enregistrement par méta-fichier existant dans la base de données.

```
FILE-CONTROL

     SELECT FFILE ASSIGN TO DSK
     ORGANIZATION IS INDEXED
     ACCESS MODE IS RANDOM
     RECORD KEY IS FFILE-SSNAME
     RECORDING MODE IS ASCII.


FILE SECTION

     FD  FFILE
          LABEL RECORD IS STANDARD
          BLOCK CONTAINS 128 RECORDS
          VALUE OF IDENTIFICATION IS "MFILESIDX".

     01  M-FILE-LINE.
         02 FFILE-PNTR       PIC 9999.
            pointeur-particularités
             si pas utilisé : 0
             sinon contient FPART-NORDRE de l'enregistrement
                   M-PART-LINE associé
         02 FFILE-COARA      PIC 99.
            numéro identifiant
         02 FFILE-SSNAME     PIC X(30).
            nom identifiant
         02 FFILE-TEXT       PIC X(49).
            contient le méta-article BD
            voir le chapitre 3
```

4.3. FICHIER DES META-OBJETS

Chaque enregistrement de ce fichier contient un article de la méta-base de données. Pour un méta-fichier donnéet un type de méta-article donné les articles doivent être trié sur leur nom s'il en ont un.

```
FILE-CONTROL

     SELECT FOBJ ASSIGN TO DSK
     ORGANIZATION IS INDEXED
     ACCESS MODE IS SEQUENTIAL
     RECORD-KEY IS FOBJ-KEY
     RECORDING MODE IS ASCII.


FILE SECTION

     FD  FOBJ
          LABEL RECORD IS STANDARD
          BLOCK CONTAINS 64 RECORDS
          VALUE OF IDENTIFICATION IS "FOBJETIDX".

     01  M-OBJET-LINE-100.
         02 FOBJ-KEY.
          03 FOBJ-SSCHEMA-IDEN      PIC 99.
           cette valeur est identique à FFILE-COARA
            du méta-fichier associé
          03 FOBJ-TYPE-IDEN         PIC 99.
           code du type de méta-article (cf. 2.2.1)
          03 FOBJ-NORDE             PIC 999.
           numéro d'ordre des articles à
           l'intérieur d'un type
         02 FOBJ-TEXT               PIC X(93).
           enrégistrement proprement dit d'un
           méta-article, pour la description voir
           le chapitre 3
```

4.4. FICHIER DES META-CHEMINS

Ce fichier contient un enregistrement par cible dans un méta-chemin d'accès.

```
FILE-CONTROL

     SELECT FCHE ASSIGN TO DSK
     ORGANIZATION IS INDEXED
     ACCESS MODE IS SEQUENTIAL
     RECORD-KEY IS FCHE-KEY
     RECORDING MODE IS ASCII.


FILE SECTION

     FD  FCHE
          LABEL RECORD IS STANDARD
          BLOCK CONTAINS 32 RECORDS
          VALUE OF IDENTIFICATION IS "FCHEMIIDX".

     01  M-CHEMIN-LINE.
         02 FCHE-KEY.
          03 FCHE-SSCHEMA-IDEN    PIC 99.
           contient le FFILE-COARA du méta-fichier
           associé
          03 FCHE-MAP-NBER        PIC 99.
           contient l'identification du méta-chemin
           d'accès (cf. 2.2.2.)
          03 FCHE-MO-NBER         PIC 999.
           contient le FOBJ-NORDE du méta-article origine
          03 FCHE-ORDER-NBER      PIC 999.
           contient le numéro d'ordre d'accès dans le chemin
         02 FCHE-MT-NBER          PIC 999.
           contient le FOBJ-NORDE du méta-article cible
```

4.5. FICHIER DES PARTICULARITES

Un enregistrement de ce fichier se rapporte à un seul méta-article qui a des particularités.

```
FILE-CONTROL

     SELECT FPART ASSIGN TO DSK
     ORGANIZATION IS INDEXED
     ACCESS MODE IS SEQUENTIAL
     RECORD-KEY IS FPART-KEY
     RECORING MODE IS ASCII.


FILE SECTION

     FD  FPART
          LABEL RECORD IS STANDARD
          BLOCK CONTAINS 109 RECORDS
          VALUE OF IDENTIFICATION IS "MPARTIIDX".

     01  M-PART-LINE.
         02 FPART-KEY.
          03 FPART-SSCHEMA-IDEN    PIC 99.
           contient le FFILE-COARA du méta-fichier
           associé
          03 FPART-NORDRE          PIC 9999.
           numéro d'ordre, correspondant au xx-PNTR
           du méta-article associé
         02 FPART-TEXT.
          03 FPART-NUM-TEXT        PIC 9(15).
          03 FPART-NUM-TEXT        PIC 9(15).
          03 FPART-ALP-TEXT        PIC X(90).
          voir le chapitre 3
```

MANUEL D' UTILISATION DU

GENERATEUR " GENER "

Juillet 1982

TABLE DES MATIERES

1. INTRODUCTION

Ce manuel suppose que le lecteur a pris connaissance du chapitre 5 du mêmoire et par ce fait même on ne peut espérer une complète compréhension de ce manuel que si l'on a lu ce chapitre.

Après l'introduction , le chapitre 2 contiendra la description du langage de commande de génération GECOL2 (GEneration COmmand Language 2 *).

Puisque le générateur travaille comme un interprêteur sur un texte non vérifié, il peut y avoir des erreurs lors de l'exécution d'un texte de commande de génération. A ce moment le générateur renvoie un message d'erreur et s'arrête. Dans le chapitre 4 le lecteur trouvera des explications supplémentaires pour chaque type d'erreur à chaque fois qu'une telle explication semble nécessaire.

Les zones mêmoires que le générateur peut utiliser sont déterminées a priori. Or ceci limite les capacités du générateur. Dans le chapitre 4 on trouvera des informations précises pour adapter les tailles des piles aux besoins de l'utilisateur.

Enfin dans le chapitre 5 sera exposée la façon de faire tourner le générateur sur TOPS-20.

---

*Ce langage est une version améliorée de GECOL que nous avons développé chez NCR.

2. LE LANGAGE DE COMMANDE DE GENERATION

2.1. INTRODUCTION

Le langage de commande de gènèration GECOL2 (GEneration COmmand Language) est un langage basé sur la notion de ligne. Chaque ligne dans un texte GECOL2 est

soit une ligne de commande de gènèration

soit une ligne de directive

soit une ligne de commentaire.

Le type d'une ligne est dèterminé par le caractère contenu dans la premiére colonne de la ligne. Une ligne de commentaire commence toujours par le caractère spècial "!", alors qu'une ligne de directive commence toujours par le caractère spècial "@". Une ligne de commande de gènèration peut commencer toujours avec n'importe quel caractère sauf les caractères "!" et "@". Le premier caractère d'une ligne de commande de gènèration fait partie du texte à recopier. Le caractère spècial "^" ne peut pas du tout être employé.

Une ligne comprend au maximum 66 caractères.

2.2. LES PARAMETRES

Un programme GECOL2 peut rèfèrencer des donnèes contenues dans la mèta-base de donnèes ou des variables de gestion à l'aide de paramètres. Nous distinguons trois types de paramètres.

1. Les paramètres de la base de donnèes qui rèfèrencent une donnèe de la mèta-base de donnèes. La liste complète de ces paramètres et leur signification sont donnèes à la fin de cette partie.

2. Les paramètres de gestion alphanumèriques permettent à un programme GECOL2 de conserver un certain nombre d'informations afin de les utiliser plus tard. Le nombre de ces variables dépend de la version du gènèrateur. En fin de cette partie seront indiquès le nombre et les identifications de ces paramètres.

3. Les paramètres de gestion numèriques permettent à un programme GECOL2 de conserver de l'information numèrique et de faire des opèrations de calcul dessus.Le nombre de ces variables dépend de la version du gènèrateur. En fin de cette partie seront indiquès le nombre et les identifications de ces paramètres.

Le dèbut d'un paramètre dans un texte GECOL2 est marquè par la prèsence du caractère spècial "#". Si l'on veut utiliser ce caractère

à d'autres fins, il doit être dédoublé. Alors chaque caractère "##" dans une commande de génération sera remplacé par un simple "#" dans le texte généré.

La longueur de l'identificateur du paramètre qui suit le caractère spécial "#" est toujours de 6 positions.

Pour les paramètres de gestion (numériques ou alphanumériques) il ne peut y avoir une ambiguité sur la valeur pointée par l'identifiant. Ainsi aucune information supplémentaire ne doit être fournie au générateur pour qu'il puisse faire le remplacement. Ces paramètres peuvent être référencés à n'importe quel moment du programme GECOL2, à condition qu'ils aient un format compatible. Le programmeur d'un texte GECOL2 veillera à ce que ces variables soient bien initialisées.

L'utilisation des paramètres-bd demande plus de précautions. D'abord pour qu'un tel paramètre puisse être utilisé, il faut qu'un méta-article contenant une donnée référencée par ce paramètre ait été accédé dans une des boucles courantes. Le paramètre sera alors remplacé par cette donnée du méta-article courant. Cependant dans certains cas un même type de méta-article peut être accédé plusieurs fois dans des boucles imbriqués. Par exemple :

```
@FOR-EACH FICHIER DO     (1)
 ....
@FOR-EACH FICHIER DO     (2)
 ....                    (a)
@OD
 ....
@OD
```

Ainsi on ne saurait dire si la donnée ratachée a un FICHIER référencée en (a) se rapporte au méta-article du FICHIER accédé dans la boucle (1) ou dans la boucle (2). Pour lever cette ambiguité, on peut faire suivre le paramètre de "n" ou n doit être un entier strictement positif. Ce nombre permet de déterminer si la référence se rapporte à la dernière, avant-dernière ... donnée accédée, répondant à référence et qui est encore courante. La dernière correspond à n=1, l'avant-dernière à n=2, etc. Dans l'exemple si l'on a en (a) :

#FIIDEN1 (FIIDEN étant l'identifiant du fichier), cela veut dire qu' on va prendre l'identification du fichier accédé en (2), alors que #FIIDEN2 veut dire qu'on se rapporte au fichier accédé en (1).

Par défaut on se rapportera toujours à la dernière donnée accédée. Ainsi #FIIDEN sera interprété comme #FIIDEN1.

Si l'on veut utiliser ce caractère "$" à d'autres fins, il doit être dédoublé. Alors chaque double caractère "$$" dans une commande de génération sera remplacé par un simple "$" dans l' interface générée.

Ce que nous avons dit sur le problème d'ambiguité est particulièrement vrai pour les paramètres référençant des données des particularités rattachées a un méta-article. Comme les données de particularités portent le même nom pour chaque type de méta-article, le nombre n tiendra compte de chaque boucle dont il faudra sortir pour

trouver la bonne valeur et non pas seulement des boucles qui se rèfèrent à un type de mèta-article particulier.

2.3. DEFINITIONS

Un littèral est un chaine de 1 à 30 caractères alphanumèriques entourèe de simples cotes (´).

Un numèrique est une chaine de 1 à 10 caractères numèriques non entourèe de cotes.

Un identifiant est une chaine de caractères composèe du signe spècial "#", du nom d´un nom d´ un paramètre plus èventuellement le caractère spècial "$", suivi d´un numèrique et du signe spècial "$".

2.4. LES DIRECTIVES

2.4.1. LE SYMBOLISME UTILISE

Le symbolisme utilisè dans ce manuel pour dècrire les directives est fortement rattachè au symbolisme utilisè dans la desciption des "statements" de COBOL.

1. Les mots reservès sont ècrits en majuscules. Tous les mot rèservèes obligatoires dans une directive sont soulignès.

2. La partie de texte du format gènèral entourèe de crochets, [], peut être omise selon l´action que l´utilisateur veut avoir faite.

3. Lorsque des parenthèses, {}, entourent une partie de texte, il faut choisir une des options offertes.

4. Les signes, ‹›, entourent un groupe qui sera explicitè ailleurs.

2.4.2. STRUCTURE D´ UN TEXTE GECOL2

Un texte GECOL2 a la structure suivante :

[commentaire]

begin directive

‹ GECOL2-text-body ›

end-directive

[commentaire]

ou

‹ GECOL2-text-body › peut comprendre :

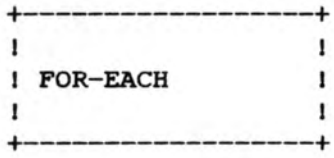
-des directives sauf les directives de dèbut et de fin de texte

-du commentaire

-des commandes de gènèration

2.4.3. STRUCTURE D´ UNE DIRECTIVE

Une directive ne peut apparaître que sur une ligne de directive, c´est-à-dire une ligne qui commence par "@". Une directive peut se rèpartir sur plusieurs lignes de directives. Cependant sur une même ligne-directive ne peuvent apparaître que des parties d´une seule directive. Deux lignes-directives comprenant une même directive peuvent être separèes par une ou plusieurs lignes de commentaire. Un mot d´ une directive doit toujours apparaître sur une seule ligne. Deux mots d´une directive doivent être sèparès par un blanc au moins. Le premier mot d´une directive peut dèbuter à partir de la deuxième position de la ligne, la première étant rèservèe au caractère spècial "@".



2.4.4. DIRECTIVE DE BOUCLE

Fonction

Cette primitive remplit une double fonction. D'une part elle permet de boucler sur une partie du texte GECOL2 et d'autre part elle rend accessible un méta-article.

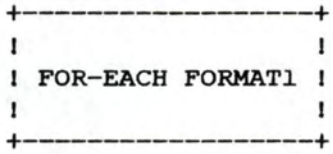
Format général

FOR-EACH type de méta-article

[{ WITHIN type de méta-chemin d' accès
  { USING type de méta-clé EQUAL { littéral-1 } } DO
                                 { identifiant-1 }
]

Notes techniques

1. Les boucles peuvent être imbriquées, mais le nombre d' imbrications maximal est limité par le générateur.



2.4.4.1. DIRECTIVE DE BOUCLE - FORMAT1

Fonction

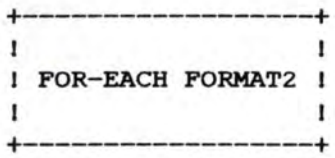
Cette directive rend accessible tous les méta-articles d'un type pour le sous-schéma courant et exécute le corps de la boucle pour chacun d'eux.

Format général

FOR-EACH type de méta-article DO

Notes techniques

1. "type de méta-article" est l'identifiant d'un type d'article de la méta-base de données. Seuls ceux-ci peuvent figurer ici. Tous les types de méta-article valables ainsi que leurs identifications exactes sont donnés en fin de cette partie.



2.4.4.2. DIRECTIVE DE BOUCLE - FORMAT2

Fonction

Cette directive rend accessible tous les méta-articles d'un type dans un méta-chemin d'accès du sous-schéma courant et exécute le corps de la boucle pour chacun d'eux.

Format général

FOR-EACH type de méta-article

WITHIN type de méta-chemin d'accès DO

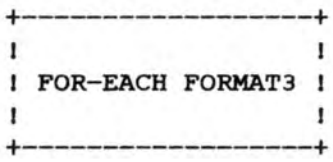
Notes techniques

1. "type de méta-article" est l'identifiant d'un type d'article de la méta-base de données. Seuls ceux-ci peuvent figurer ici. Tous les types d'article valables ainsi que leurs identifications sont donnés en fin de cette partie.

2. "type de méta-chemin d'accès" est l'identifiant d'un type de chemin d'accès de la méta-base de données. Seuls ceux-ci peuvent figurer ici. Tous les types de chemin valables ainsi que leurs identifications sont donnés en fin de cette partie.

3. Le type de méta-article doit être un type de cible valable pour le type de méta-chemin d'accès.

4. Il faut que dans une boucle extérieure un méta-article du type origine pour que le méta-chemin soit accédé. Si plusieurs méta-articles du type de l'orgine sont courants, le dernier accédé sera considéré comme origine.



2.4.4.3. DIRECTIVE DE BOUCLE - FORMAT3

Fonction

Cette directive rend accessible tous les méta-articles d'un type du sous-schéma courant qui vérifient une valeur de clé et exécute le corps de la boucle pour chacun d'eux.

Format général

FOR-EACH type de méta-article USING type de méta-clé

EQUAL { littéral-1 / identifiant } DO

Notes techniques

1. "type de méta-article" est l'identifiant d'un type d'article de la méta-base de données. Seuls ceux-ci peuvent figurer ici. Tous les types d'article valables ainsi que leurs identifications sont donnés en fin de cette partie.

2. "type de méta-clé" est l'identifiant d'un type de clé d'accès de la méta-base de données. Seuls ceux-ci peuvent figurer ici. Tous les types de clé valables, leurs identifications ainsi que le type de méta-article auquel chaque clé se rattache seront données en fin de cette partie.

3. Il faut que le type de méta-clé soit un type valable pour le type de méta-clé.

4. littéral-1 et identifiant-1 respectent les définitions faites en 2.3. .

5. identifiant-1 peut être des trois types

2.4.5. DIRECTIVE DE FIN DE BOUCLE

Fonction

Cette directive limite le corps de la boucle la plus intérieure non encore fermée.

Format général

OD

Notes techniques

1. Il faut que toute sélection intérieure à cette boucle soit terminée.

2.4.6. DIRECTIVE DE SORTIE DE BOUCLE

Fonction

Cette directive permet la sortie prématurée d'une ou plusieurs boucles imbriquées.

Format général

BREAK[$numérique-1$]

Notes techniques

1. Si "BREAK" apparaît seul, on sort d'une seule boucle, la boucle courante la plus intérieure.

2. numérique-1 doit respecter les considérations faites en 2.3. .

3. numérique-1 indique le nombre de boucles dont il faut sortir. Les boucles sont comptées à partir de la boucle la plus intérieure.

2.4.7. DIRECTIVE DE SELECTION

Fonction

Cette directive permet de déterminer s'il faut ou non exécuter une partie du texte GECOL2, le corps de la sélection, et cela en fonction des valeurs qu'ont pris certaines données, à ce moment précis de l'exécution.

Format général

IF < condition > THEN

Notes techniques

1. Les sélections peuvent être imbriquées, mais le nombre maximal d'imbrications est imposé par le générateur.

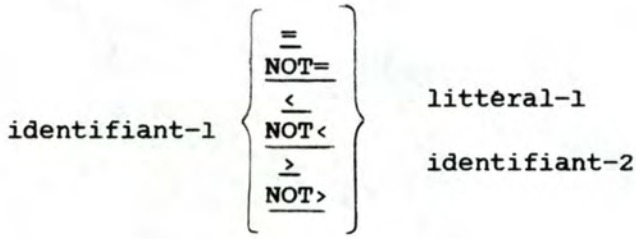
2. Si le résultat de l'évaluation de la condition est "TRUE" le corps de la sélection sera exécuté; si le résultat est "FAUX" il ne sera pas exécuté.

2.4.7.1. CONDITION DE SELECTION

La condition est constituée d'une ou plusieurs conditions élémentaires qui sont combinées à l'aide de OU, ET et NON logiques ainsi qu'avec une structure de parenthèses.

a) la condition élémentaire

Format général



Notes techniques

1. littéral-1, identifiant-1, identifiant-2 doivent respecter les définitions données en 2.3. .
2. identifiant-1 et identifiant-2 peuvent être des 3 types.
3. si deux valeurs numériques sont comparées, on fait un test numérique sinon un test alphanumérique.

b) les possibilités de combinaison

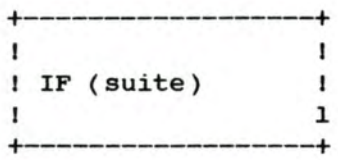
Format

le OU logique : OR

le ET logique : AND

le NON logique : NOT

les parenthèses: ( )



Notes techniques

1. L'évaluation de la condition se fait de gauche à droite sans priorité sur les opérateurs OR, AND et NOT.

2. Les parenthèses permettent de faire évaluer des résultats partielles.

3. Les parenthèses peuvent être imbriquées.

4. La complexité de la condition admise est limitée par le générateur selon la taille de la pile des opérateurs non encore appliqués et la taille de la pile des résultats partiels non encore appliqués.

5. La composition de la condition élémentaire doit respecter l'ordre décrit ci-dessous.

les états :
1 début de la condition
2 condition élémentaire
3 AND
4 OR
5 (
6)
7 NOT
8 fin de la condition

les combinaisons permises (x = permise)

previous state \ next state	1	2	3	4	5	6	7	8
1		x			x		x	
2			x	x		x		x
3		x			x		x	
4		x			x		x	
5		x			x		x	
6			x	x		x		x
7					x			

en plus il faut vérifier que les parenthèses ouvrantes et les parenthèses fermantes soient bien mises.

2.4.8. DIRECTIVE DE FIN DE SELECTION

Fonction

Cette directive limite le corps de la sélection la plus intérieure non encore fermée.

Format général

FI

Notes techniques

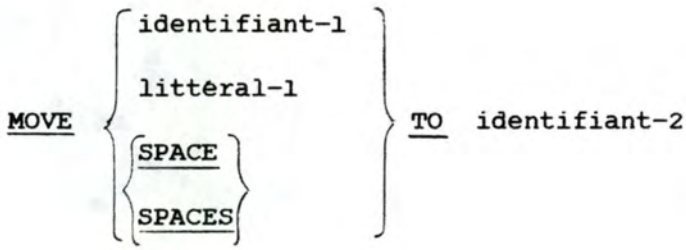
1. Il faut que toute boucle intérieure à cette sélection soit terminée.

2.4.9. DIRECTIVE DE GARNISSAGE

Fonction

Cette directive permet d'affecter une valeur à une des variables de gestion.

Format général



Notes techniques

1. littéral-1, identifiant-1, identifiant-2 doivent respecter les définitions faites en 2.3.
2. identifiant-1 peut être des trois types
3. identifiant-2 ne peut pas être un paramètre de base de données
4. l'option SPACE/SPACES permet de mettre une zone à blanc

2.4.10. DIRECTIVE D' ADDITION

Fonction

Cette directive permet d'additionner deux valeurs dans une variable de gestion numérique.

Format général

```
ADD { numérique-1   } TO { numérique-2   }
    { identifiant-1 }    { identifiant-2 }

    GIVING identifiant-3
```

Notes techniques

1. numérique-1, numérique-2, identifiant-1, identifiant-2 et identifiant-3 doivent respecter les définitions faites en 2.3.

2. identifiant-1 et identifiant-2 peuvent être des trois types. Il doivent contenir dans tous les cas une valeur numérique.

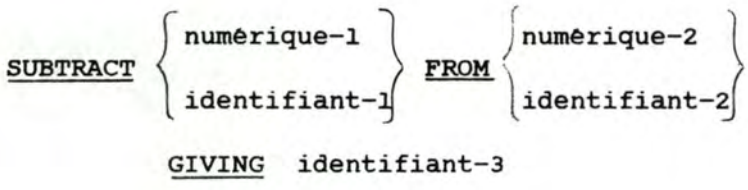
3. identifiant-3 ne peut être qu'une variable de gestion numérique

2.4.11. DIRECTIVE DE SOUSTRACTION

Fonction

Cette directive permet de soustraire une valeur d'une autre; le résultat étant placé dans une variable de gestion numérique.

Format général



Notes techniques

1. numérique-1, numérique-2, identifiant-1, identifiant-2 et identifiant-3 respectent les définitions énoncées en 2.3. .
2. identifiant-1 et identifiant-2 peuvent être des trois types. Ils doivent cependant contenir dans tous les cas des valeurs numériques.
3. identifiant-3 ne peut être qu'une variable de gestion numérique.

2.4.12 DIRECTIVE DE DEBUT

Fonction

Cette directive doit signaler le début de corps d'un texte GECOL2.

Format général

BEGIN

2.4.13 DIRECTIVE DE FIN

Fonction

Cette directive doit signaler la fin du corps du texte GECOL2.

Format général

END

2.5. LES COMMANDES DE GENERATION

2.5.1. GENERALITES

Chaque ligne de commande de génération est considérée comme étant une commande de génération. Une ligne de commande de génération peut apparaître n'importe où dans le corps d'un texte de génération. La seule limitation est qu'elle ne peut pas couper en deux une directive.

Une telle ligne peut débuter par tout caractère sauf les caractères spéciaux "@" et "!". Le premier caractère fait déjà partie du texte à générer.

Pour les textes COBOL à générer, il faut remarquer que la colonne 1 d'une ligne de commande de génération correspond à la colonne 7 d'une ligne du programme généré; le numéro de ligne étant produit automatiquement par le générateur à partir de 0 avec un incrément de 1.

2.5.2. LIGNE GENEREE

Une ligne générée se veut une copie aussi fidèle que possible d'une ligne en entrée. Le recopiage se fait caractère par caractère de la ligne en entrée vers la ligne en sortie, en sachant que la colonne 1 de la commande de génération correspond à la colonne 1 de la ligne générée. Si un paramètre apparaît dans une ligne de commande de génération il est remplacé comme décrit au chapitre 5 du mémoire. A partir de ce moment il y a un décalage dans la ligne générée par rapport à la ligne en entrée.

2.5.3. DEBORDEMENT

Ce décalage dans la ligne générée peut entraîner un débordement sur l'enregistrement contenant cette ligne. La prise en charge des débordements envisagée dans la version courante du générateur se rapporte uniquement aux textes COBOL. En plus le générateur applique uniquement les règles de coupure et de continuation appliquables aux mots et aux littéraux numériques. La coupure est donc mal faite dans tous les autres cas.

Dans le texte généré est ajoutée une ligne de commentaire pour avertir l'utilisateur qu'il y a un débordement. L'utilisateur est prié de vérifier que la coupure a été bien faite.

En plus il faut remarquer que les positions de 73 à 80 d'une ligne COBOL ne sont pas utilisables ici; cela poserait trop de problèmes de déplacement dans la ligne.

2.6. COMMENTAIRE

Une ou plusieurs lignes de commentaire peuvent apparaître à n'importe quel moment sur le texte GECOL2. Une telle ligne a obligatoirement le caractère "!" en première colonne.

Il faut remarquer que le commentaire qui doit être généré sur le texte en sortie peut aussi commenter les macros. Un tel commentaire est à traiter comme n'importe quel autre texte à générer.

2.7. TYPES DE META-ARTICLE

Dans cette partie sont énoncés les types de méta-article valables pour cette version du générateur et qui peuvent donc être utilisés dans les directives. Les méta-artilces qu'on peut référencer sont ceux définis par le modèle d'accès généralisé.

BD

description de la base de données

FICHIER

description d'un fichier

TARTICLE

description d'un type d'article

TCHEMIN

description d'un type de chemin d'accès

ITEM

description d'un item

ORIGINE

description d'une origine

CIBLE

description d'une cible

GLOBAL

description d'un global

SIMPLE

description d'un simple

COMPOSANT

description d'un composant

2.8 TYPES DE META-CHEMINS D'ACCES

Sont repris ici tous les types de méta-chemins valables pour cette version du générateur.

BDSI

de bd vers simple

FIGL

de fichier vers global

FISI

de fichier vers simple

TASI

de tarticle vers simple

TAFI

de tartilce vers fichier

TAOR

de tarticle vers origine

TACI

de tarticle vers cible

TAIT

de tarticle vers item

TASI

de tarticle vers simple

TCOR

de tchemin vers origine

TCCI

de tchemin vers cible

TCGL

de tchemin vers global

TCSI

de tchemin vers simple

TCCO

de tchemin vers composant

ITCO

de item vers composant

SICO

de simple vers composant

ITIT

de item vers item

GLSI

de global vers simple

2.9 TYPES DE META-CLES D'ACCES

Ici seront énoncées toutes les clès valables pour cette version du générateur. En outre il sera précisé à quel type de méta-article ils se rattachent.

BDCODE

code de la base de données

BDIDEN

identification de la base de données

FICODE

code du fichier

FIIDEN

identification du fichier

TACODE

code de tarticle

TAIDEN

identifiation de tarticle

TCCODE

code de tchemin

TCIDEN

identification de tchemin

ITCODE

code de item

ITIDEN

identification de item

GLCODE

code de global

SICODE

code de simple

2.10. LISTE DES PARAMETRES

a) paramètres-bd

Ces paramètres sont définis en détail dans le chapitre 4.2. du mémoire. Ainsi l'utilisateur ne trouvera qu'une description sommaire et incomplète de ces paramètres.

BDCODE: code de la bd
BDIDEN: nom de la bd
BDINTE: nom de l'interface
BDLGMX: longueur du plus grand type d'articles de cette bd
BDNBFI: nombre de fichiers dans la bd
BDNBRE: nombre de types d'articles dans la bd
BDNBSE: nombre de types de chemins d'accès
BDOP01: indicateur si une operation est permis ou non
BDOP02: "
BDOP03: "
BDOP04: "
BDPASW: mot de passe de la bd
BDPNTR: pointeur particularités pour bd

CICOTA: code du tarticle qui est cible
CICOTC: code du tchemin auquel se rapporte la cible
CIMINS: mode d'insertion de la cible
CIMRET: mode de retrait d'une cible du chemin d'accès
CIOP01: indicateur si une opération est permis ou non
CIOP02: "
CIOP03: "
CIOP04: "
CIPNTR: pointeur particularités pour cible

COCOSI: code du domaine simple associé
COITTC: indique si un composant est associé à un
item ou a un tchemin
COPNTR: pointeur particularités du composant
COSENS: détermine le sens de l'ordre sur un item
COTYPE: type du composant

FICODE: code du fichier
FIIDEN: identification du fichier
FINBTA: nombre de tarticle dans le fichier
FIOP01: indicateur si une opération est permis ou non
FIOP02: "
FIOP03: "
FIOP04: "
FIOP05: "
FIOP06: "
FIOP07: "
FIOP08: "
FIPNTR: pointeur particularités pour le fichier

GLCLID: indicateur si global est un identifiant

GLCLKE: indicateur si global est une clé
GLCLOR: indicateur si global est un ordre
GLCODE: code du global
GLDBLE: détermine la facon de traiter les doubles
GLORDR: détermine l'ordre retenu
GLPNTR: pointeur particularités pour le global
GLREFE: référentiel retenu
GLTYRF: type du référentiel retenu

ITCMPT: compteur de répétitivité de l'item
ITCOAS: code du tarticle associé
ITCODE: code de l'item relatif à un tarticle
ITCOIT: code de l'item composant
ITCOTA: code du tarticle auquel l'item est rattaché
ITDECI: position du point décimal
ITIDEN: identification de l'item
ITLONG: longueur de l'item
ITNBIT: nombre d'items associés
ITNBCO: nombre de composants associés
ITNOLV: numéro de niveau
ITOBFA: indicateur obligatoire/facultatif
ITOP01: indicateur si une opération est permis ou non
ITOP02: "
ITPNTR: pointeur particularités de item
ITRPMX: répétition maximale
ITSTRU: type de la structure interne
ITUNIT: type de code interne utilisé

ORCOTA: code de tarticle qui est origine
ORCOTC: code du tchemin d'accès auquel elle se rapporte
ORMINS: détermine le mode d'insertion
ORMRET: mode de retrait de l'origine
OROP01: indicateur si une opération est permis ou non
OROP02: "
OROP03: "
OROP04: "
ORPNTR: pointeur particularités de l'origine

PAALH1: zone alphanumérique dans particularités
PAALH2: "
PAALH3: "
PANUM1: zone numérique dans particularités
PANUM2: "
PANUM3: "
PANUM4: "
PANUM5: "

SICLID: indique si simple est un identifiant
SICLKE: indique si simple est une clé
SICLOR: indique si simple est un ordre
SICODE: code de simple
SICOTA: tarticle auquel simple est rattaché
SIDBLE: détermine le traitement des doubles
SIORDE: détemine le type d'ordre
SIPNTR: pointeur particularités de simple
SIREFE: référentiel du simple
SITYRF: type du référentiel

TACODE: code de tarticle
TAIDEN: identification de tarticle
TALONG: longueur du type d'article
TANBCI: nombre de fois qu'il est cible
TANBFI: nombre de fichier auxquels il appartient
TANBIT: nombre d'items qu'il a
TANBOR: nombre de fois qu'il est origine
TANBSI: nombre de simples associés à ce tarticle
TAOP01: indicateur si une opération est permise ou non
TAOP02: "
TAOP03: "
TAOP04: "
TAOP05: "
TAPNTR: pointeur particularités de tarticle

TCCODE: code du type de chemin d'accès
TCCONN: connectivité du tchemin
TCIDEN: identification du tchemin
TCINVE: code identifiant du tchemin inverse
TCNBCI: nombre de type de cibles
TCNBCO: nombre de composants assosiés
TCNBGL: nombre de globalas associés
TCNBOR: nombre de type d'origine
TCNBSI: nombre de simples associés
TCPNTR: pointeur particularités de tchemin

b) les variables de gestion

Les variables de gestion alphanumériques sont identifiées par ALPHnn, où nn est un nombre de 01 à xx, et où xx dépend de la version du générateur. La valeur pour xx est indiquée en 4.5..

Les variables de gestion numériques sont identifiées par NUMEnn, où nn est un nombre de 01 à yy,et où yy dépend de la version du générateur. La valeur pour yy est indiquée en 4.5..

3. LES ERREURS

3.1. INTRODUCTION

Comme nous l'avons dit dans l'introduction de ce manuel, les erreurs dans un programme GECOL2 sont détectées à l'exécution de ce programme par le générateur. Si ce dernier découvre une erreur dans le texte, il s'arrête directement en signalant le type d'erreur et l'endroit dans le programme GECOL2 où cette erreur est apparue. Dans la partie suivante nous reprenons les messages d'erreur qui peuvent apparaître et s'il s'avère nécessaire, nous donnerons des indications pour résoudre le problème.

3.2 LES ERREURS POSSIBLES

1. DB CAN'T BE OPENED: code de retour

2. SUBSCHEMA CAN'T BE ACCESSED: code de retour

3. SUBSCHEMA CAN'T BE FREED: code de retour

4. DB CAN'T BE CLOSED: code de retour

   Dans ces quatre cas le générateur a détecté un problème au niveau de la base de données des schémas. A l'aide du code de retour que cette dernière renvoie, l'utlisateur peut détecter le genre de problème qui s'est posé en consultant le manuel de l'utilisateur de la méta-base de données.

5. UNEXPECTED DB-RESPONSE: code de retour

   Dans ce cas il y a aussi un problème au niveau de la méta-base de données. La réponse inattendue a été fournie lors de la consultation d'un méta-article suite à l'exécution d'une boucle.

6. RECORD TYPE UNKNOWN: mot

7. ACCESS-PATH-TYPE UNKNOWN: mot

8. ACCESS-KEY-TYPE UNKNOWN: mot

   Ces trois erreurs sont détectées lors de l'interprétation d'un FOR-EACH. L'utilisateur a utilisé un identifiant d'un type qui est inconnu au générateur. Les identifiants des types permis sont données en 2.7., 2.8. et 2.9..

9. UNKNOWN PARAM: param

10. UNKNOWN INTERNAL VARIABLE USED

Ces deux erreurs peuvent intervenir à n'importe quel moment de l'interprétation du texte GECOL2. L'identifiant du paramètre utilisé est inconnu au générateur. Les identifiants permis sont donnés en 2.10. de ce manuel. S'il y a un problème au niveau des variables internes, cela peut être dû au fait qu'il n'y en a pas assez. Si l'on veut étendre ce nombre, on est prié de consulter le sous-chapitre 4.5. de ce manuel.

11. TOO MUCH IMBRICATED LOOPS/SELECTIONS

12. CAPACITY OF GENERATOR EXCEEDED

Dans ces deux cas-ci l'erreur provient du fait que les zones mémoires du générateur sont insuffisantes pour traiter le problème. L'utilisateur est prié de se référer au chapitre 4 de ce manuel afin de pouvoir adapter le générateur à ses besoins.

13. NOT ALL FOR-EACH-LOOPS ARE CLOSED AT END

14. NOT ALL IF ARE CLOSED AT END

15. NO CORRESPONDING 'FOR-EACH' WAS FOUND FOR THE 'OD' AT LINE:

16. NO CORRESPONDING 'IF' WAS FOUND FOR THE 'FI' AT LINE:

L'utilsateur doit vérifier que les fermetures des boucles et des sélections sont bien faites.

17. OVERLAPPING OF A LOOP AND A SELECTION

Une boucle doit être entièrement comprise dans une sélection ou inversément une sélection doit être entièrement comprise dans une boucle.

18. ORIGIN MISSING FOR ACCESS-PATH

Cette erreur est détectée lors de l'interprétation du FOR-EACH format2. Il n'y a pas de méta-article courant du type de l'origine. Il faut assurer qu'un tel article soit accédé dans une boucle extérieure.

19. THE BREAK-NUMBER IS MORE IMPORTANT THAN THE NUMBER OF CURRENT LOOPS

20. THE EXIT-NUMBER number AFTER param IS INCOMPATIBLE

Dans ces deux cas le générateur se voit dans l'impossibilité de répondre à la demande formulée, car il n'y a pas assez de boucles courantes qui satisfont les conditions.

21. VALUES MAY ONLY BE TRANSFERRED TO INTERNAL VAR

Ici l'utilisateur essaie de garnir une zone qui n'est pas une variable de gestion. Or ceci n'est pas permis.

22. NON-NUMERIC DETECTED WHERE A NUMERIC WAS EXPECTED

Deux explications sont possibles. Soit l'utilsateur essaie de garnir une variable de gestion numérique avec un non-numérique ce qui est strictement interdit. Soit l'utilisateur a employé **ALPHxx** ou **NUMEyy** où **xx** ou **yy** ne sont pas numériques.

23. THE LAST DIRECTIVE WAS INCOMPLETE

Ici on a détecté un directive incomplète. L'erreur peut éventuellement provenir du fait qu'on a insérè une ligne de commande de génération, en particulier une ligne blanche, entre deux lignes de directive contenant des parties d'une même directive.

24. OTHERS

Les autres erreurs sont des erreurs de format. Le diagnostic est directement donné à l'utilisateur.

4. ADAPTATIONS DU GENERATEUR

4.1. NECESSITE DE CALIBRER

Le générateur a besoin d'un certain nombre de piles et de zones de mémoire pour pouvoir fonctionner. Or la taille nécessaire de certaines de ces zones et en particulier celle des piles dépend du problème posé. Mais comme le générateur est écrit en COBOL, la taille de ces zones doit être définie avant la compilation. Ainsi une option a priori a dû être prise; mais il se peut que pour un certain texte de commande GECOL2 les options prises soient insuffisantes.

Le but de ce quatrième chapitre est d'aider l'utilisateur a faire les adaptations necessaires au générateur le cas échéant. Pour chaque zone de mémoire qui peut poser des problèmes on exposera:

- utilité de la zone
- causes des problèmes
- option actuelle
- comment changer.

L'utilisateur doit faire les changements exposés ici sur le programme source du générateur et le compiler par la suite.

4.2. TABLE DE GESTION DES BOUCLES

Utilité de la table

Cette table permet au générateur d'accepter une structure de boucles imbriquées dans le texte de génération. Pour chaque boucle courante le générateur conserve dans cette table

1. le type de boucle (simple, par chemin d'accès ou par clé d'accès)
2. les informations nécessaires à l'appel de la méta-base de données, qui ont ont été collectées lors de l'analyse de la directive de boucle
3. le dernier enrégistrement fourni par la méta-base de données suite à un appel du générateur pour cette boucle.

Les informations 1) et 2) permettent au générateur de faire un nouvel appel à la méta-base de données si c'est nécessaire. Les informations 3) peuvent être utilisées pour l'exécution d'une directive ou d'une commande de génération dans le corps de la boucle.

Causes des problèmes

Pour chaque boucle courante il y a une ligne prévue dans la pile. Or le nombre maximal de lignes est fixé. Il se peut que ce nombre soit insuffisant pour les macros qu'on veut sousmettre au générateur. Si le générateur détecte ce problème, il indiquera:

```
CAPACITY OF GENERATOR EXCEEDED
PROBLEME DETECTED AT LINE: numéro de la ligne des macros
```

Option actuelle

Actuellement 8 boucles peuvent être imbriquées

Comment changer

On veut changer les 8 boucles en n boucles. Alors les changements à faire dans le programme GENER:

1. Dans la table de gestion des boucles:

```
01  FOR-EACH-TABLE
    02 FOR-EACH-LINE OCCURS 8 --> n TIMES.
    ...
```

2. Dans les maximums:

```
01  MAXIMUMS.
    ...
    02 FE-LEVEL-MAX  PIC 99  VALUE 8 --> n .
    ...
```

Ne pas dépasser 99, car alors les changements seront plus importants.

4.3. PILE DE GESTION DES IMBRICATIONS DES BOUCLES ET DES SELECTIONS

Utilité de la pile

Cette pile permet au générateur de vérifier qu'une sélection est entièrement comprise dans une boucle ou inversément qu'une boucle est entièrement comprise dans une sélection.

Causes des problèmes

Pour chaque boucle et chaque sélection il y a une ligne prévue dans la pile. Or il y a une taille maximale fixée pour la pile. Il se peut que ce nombre soit insuffisant pour le texte GECOL2 qu'on veut sousmettre au générateur. Si le générateur détecte ce problème, il indique:

```
TOO MUCH IMBRICATED LOOPS/SELECTIONS!
ERROR DETECTED AT LINE : numéro de la ligne courante
```

Option actuelle

Actuellement 15 boucles/sélections peuvent être imbriquées.

Comment changer

On veut changer les 15 niveaux en n niveaux.

1. dans la pile de gestion des imbrications

```
01   FE-IF-TABLE.
     02 FE-IF-IND  PIC 9 OCCURS 15 --> n TIMES.
     ...
```

2. dans les maximums

```
01   MAXIMUMS.
     ...
     02 FE-IF-COMPT-MAX   PIC 99 VALUE 15 --> n.
     ...
```

Ne pas dépasser les 99 niveaux car alors les changements seraient plus importants.

4.4. PILES SERVANT A L'EVALUTION DE LA CONDITION

Utilité des piles

Une première pile sert à stocker les résultats intermédiaires obtenus à l'évaluation d'une condition dans sélection. La deuxième pile sert à mémoriser les parenthèses ouvertes non encore fermées, les NOT, OR et AND non encore appliqués aux résultats intermédiaires.

Causes des problèmes

Pour ces deux piles il y a une taille maximale fixée à priori. Cependant il se peut que la complexité d'une condition exige des piles plus importantes. Dans ce cas le générateur signale le problème par le message suivant:
CAPACITY OF GENERATOR EXCEEDED
PROBLEM DETECTED AT LINE : numéro de la ligne courante

Option actuelle

Actuellement 10 résultats intermédiaires peuvent être en attente.

Comment changer

On veut changer les 10 résultats intermédiaires en n résutats intermédiaires.

1. dans la pile stockant les résultats intermédiaires

```
01 RESULT-STACK.
   02 RESULT  PIC 9 OCCURS 10 --> n TIMES.
```

2. dans la pile stockant les opérateurs non encore appliqués

```
01  COND-STATE-STACK.
    02 COND-STATE  PIC 9 OCCURS 10 --> n TIMES.
```

3. dans les maximums

```
01  MAXIMUMS.
    ...
    02 MAX-COND-LEVEL   PIC 99  VALUE 10 --> n TIMES.
    02 MAX-RESULT-LEVEL PIC 99  VALUE 10 --> n TIMES.
```

Ne pas dépasser les 99 résultats intermédiares, car sinon les changements seront plus importants.

4.5. ADAPTATION DU NOMBRE DES VARIABLES DE GESTION NUMERIQUES ET ALPHANUMERIQUES

Utilité

Ce sont des mémoires qui sont mises à disposition du programmeur GECOL2 afin qu'il puisse y faire des sauvetages de valeurs et des calculs.

Causes des problèmes

Le nombre de zones mises à disposition peut être insuffisant.

Option actuelle

Actuellement on peut disposer de 20 zones numériques et 20 zones alphanumériques.

Comment changer

Si l'on a n zones numériques, il faut qu'elles s'appellent: NUME01, NUME02, ... NUMEn. Si l'on a m zones alphanumériques il faut qu'elles s'appellent: ALPH01, ALPH02, ... ALPHm. Il faut que m et n soient inférieurs à 100.

Si l'on veut changer les 20 zones numériques en n zones numériques

1. dans la table des variables de gestion numériques

```
01  VAR-NUME-TABLE.
    02 VAR-NUME    PIC 9(10)  OCCURS 20 --> n TIMES.
```

2. dans les maximums

```
01  MAXIMUMS.
    ...
    02 MAX-VAR-NUME-TABLE  PIC 99  VALUE 20 --> n.
    ...
```

Si l'on veut changer les 20 zones alphanumériques en m zones alphanumériques

1. dans la table des variables alphanumériques

```
01 VAR-ALPH-TABLE.
   02 VAR-ALPH   PIC X(30) OCCURS  20 --> m TIMES.
```

2. dans les maximums

```
01  MAXIMUMS.
    ...
    02 MAX-VAR-ALPH-TABLE  PIC 99 VALUE 20 --> m .
    ...
```

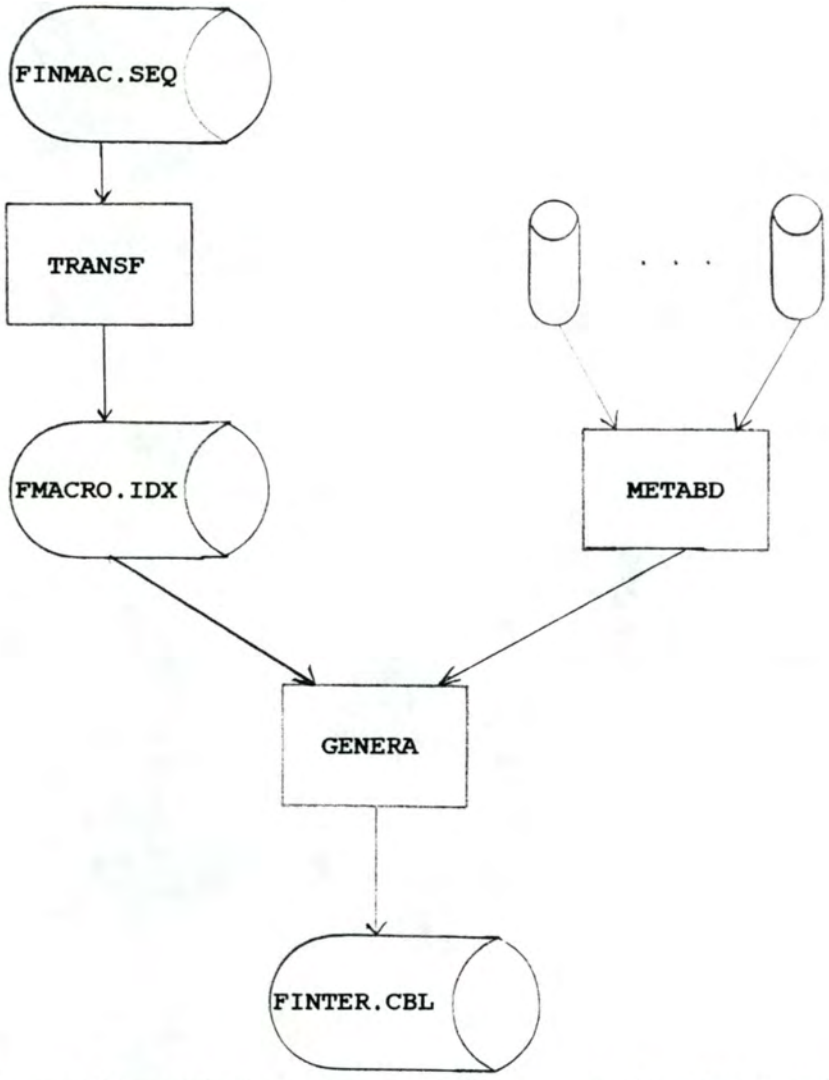
5. UTILSATION AVEC TOPS-20

5.1. INTRODUCTION

Cette partie doit aider l'utilisateur à faire une génération sur le système d'exploitation TOPS-20.

Beaucoup de choses, comme par exemple faire exécuter un programme, peut se faire de différentes manières sur TOPS-20. Cependant nous exposerons seulement une manière de faire dans chaque cas.

5.2 ARCHITECTURE



Dans cette architecture ne sont pas repris les fichiers de la méta-base de données. Nous renvoyons l'utilisateur au "Manuel d'utilisation de la méta-base de données".

5.3. LES OPERATIONS A EFFECTUER

Nous décrivons ici une suite possible des operations. L'utilisateur peut en faire une autre s'il connaît le système.

1. Si l'utilisateur a fait une adaptation du générateur, il doit recompiler celui-ci:

```
@load gener,metabd
  ...
 EXIT
@save
 GENERA saved
```

2. Pour écrire le texte de commandes de génération nous conseillons de prendre l'éditeur EMACS. L'écriture se fait dans un fichier auquel l'utilisateur donne un nom de son choix, disons XXX.SEQ.

   Après que ce texte est écrit il faut le transformer en un fichier séquentiel indexé sur le numéro de ligne. Pour faire cela, l'utilisateur doit effectuer:

```
@COPY XXX.SEQ <ESC> FINMAC.SEQ
l'utilisateur doit assurer qu'il a un fichier vide FMACRO.SEQ
@ISAM
*FMACRO,FMACRO=FMACRO.SEQ/B
Mode of input file: A
mode of output file : A
Maximum record size : 80
Key descriptor: UN1.5
Records per input block : 0
Total records per Data block (Recommended: 64):
Empty records per Data block: 0
Total entries per Index block (Recommended: 42):
Empty entries per Index block: 2
Percentage of Data file to leave empty: 0
percentage of Index file to leave empty: 0
Maximum number of records file can become: 10000
 ....
^C
@RUN TRANSF
```

3. Pour exécuter le programme:

```
RUN GENERA
```

Si la génération s'est terminée normalement, l'interface générée se trouve dans FINTER.CBL.

L´ EXEMPLE PETITPAS

août 1982

TABLE DES MATIERES

page

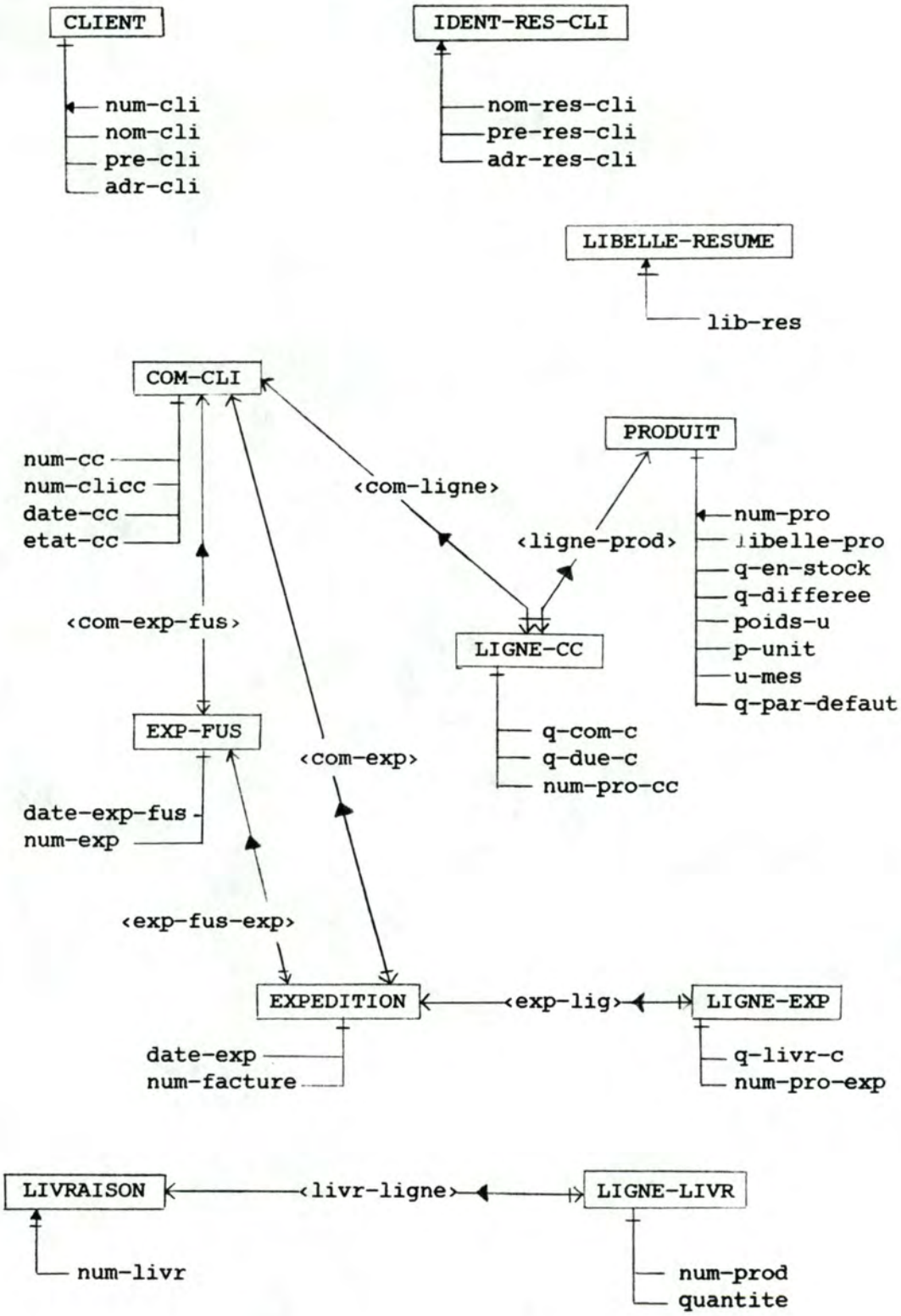
1. INTRODUCTION

Nous avons retenu la base de données du cas Petitpas comme exemple et comme base de données de test pour le système. Ici nous n'entrerons pas dans les détails du cas Petitpas. De cet exemple nous ne retenons qu'une implémentation possible de la base de données et dont le schéma d' accès est donné au deuxième chapitre. Ce schéma est décrit dans le DDL de DBMS-20 au chapitre trois.

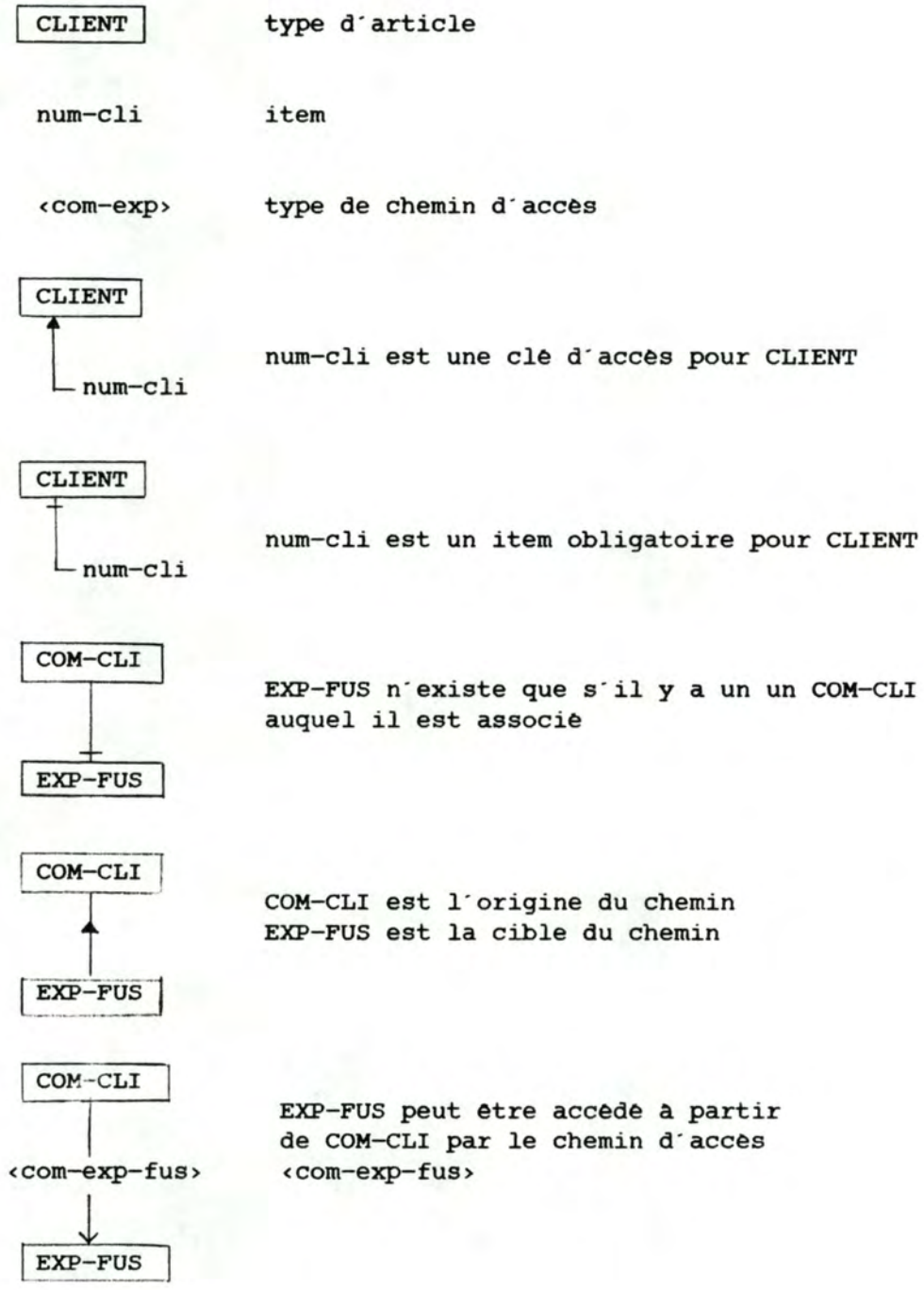
Au quatrième chapitre a été développé un petit exemple de génération qui comprend d'une part les commandes de génération et d'autre part le texte généré. Ainsi le lecteur pourra se faire une image des possibilités de génération du système.

L'interface complet pour la base de données Petitpas implémentée sur DBMS-20 est donnée dans le deuxième volume des annexes.

2. LE SCHEMA DES ACCES



La signification des symbolismes utilisés dans le schéma est la suivante:



3. DESCRIPTION DDL DU SCHEMA

```
NOTE ALL EXCEPTIONS.

ASSIGN ar-com TO a-one
        RPP 50
        CALC AT MOST 5 RPP
        FIRST PAGE 1
        LAST PAGE 20
        PAGE SIZE IS 512 WORDS.

ASSIGN ar-produit TO a-two
        RPP 15
        CALC AT MOST 5 RPP
        FIRST PAGE 30
        LAST PAGE 40
        PAGE SIZE IS 512 WORDS.

ASSIGN ar-client TO a-three
        RPP 20
        CALC AT MOST 5 RPP
        FIRST PAGE 50
        LAST PAGE 60
        PAGE SIZE IS 512 WORDS.


SCHEMA NAME IS ptpas.


AREA NAME IS ar-produit.
AREA NAME IS ar-client.
AREA NAME IS ar-com.


RECORD NAME IS client  (#1#)
        LOCATION MODE IS CALC USING num-cli (#1#)
        DUPLICATES ARE NOT ALLOWED
        WITHIN ar-client.
        02 num-cli              PIC 9(7).
        02 nom-cli              PIC X(30).
        02 pre-cli              PIC X(30).
        02 adr-cli              PIC X(30).

RECORD NAME IS ident-res-cli
        LOCATION MODE IS CALC USING nom-res-cli pre-res-cli adr-res-cli
        DUPLICATES ARE NOT ALLOWED
        WITHIN ar-client.
                02 nom-res-cli  PIC X(30).
                02 pre-res-cli  PIC X(30).
                02 adr-res-cli  PIC X(30).
```

```
RECORD NAME IS libelle-resume
        LOCATION MODE IS CALC USING lib-res
        DUPLICATES ARE NOT ALLOWED
        WITHIN ar-produit.
                02 lib-res       PIC X(30).

RECORD NAME IS com-cli  (#2#)
        LOCATION MODE IS CALC USING num-cc      (#2#)
        DUPLICATES ARE NOT ALLOWED
        WITHIN ar-com.
                02 num-cc                PIC 9(7).
                02 num-clicc             PIC 9(7).
                02 date-cc               PIC 9(6).
                02 etat-cc               PIC 9.



RECORD NAME IS produit   (#4#)
        LOCATION MODE IS CALC USING num-pro (#4#)
        DUPLICATES ARE NOT ALLOWED
        WITHIN ar-produit.
                02 num-pro               PIC 9(7).
                02 libelle-pro           PIC X(30).
                02 q-en-stock            PIC 9(7).
                02 q-differee            PIC 9(7).
                02 poids-u               PIC 9(7).
                02 p-unit                PIC 9(7).
                02 i-epuis               PIC X.
                02 u-mes                 PIC X(30).
                02 q-par-defaut          PIC 9(4).


RECORD NAME IS ligne-cc  (#3#)
        LOCATION MODE IS VIA com-ligne
        WITHIN ar-com.
                02 q-com-c               PIC 9(4).
                02 q-due-c               PIC 9(4).
                02 num-pro-cc            PIC 9(7).


RECORD NAME IS expedition
        LOCATION MODE IS VIA com-exp
        WITHIN ar-com.
                02 date-exp              PIC 9(6).
                02 num-facture           PIC 9(7).

RECORD NAME IS exp-fus
        LOCATION MODE IS VIA com-exp-fus
        WITHIN ar-com.
                02 date-exp-fus          PIC 9(6).
                02 num-exp               PIC 9(8).
```

```
RECORD NAME IS ligne-exp
        LOCATION MODE IS VIA exp-lig
        WITHIN ar-com.
                02 q-livr-c                PIC 9(4).
                02 num-pro-exp             PIC 9(7).


RECORD NAME IS livraison
        LOCATION MODE IS CALC USING num-livr
        DUPLICATES ARE NOT ALLOWED
        WITHIN ar-produit.
                02 num-livr                PIC 9(7).

RECORD NAME IS ligne-livr
        LOCATION MODE IS VIA livr-ligne
        WITHIN ar-produit.
                02 num-prod                PIC 9(7).
                02 quantite                PIC 9(7).


SET NAME IS com-ligne
        MODE IS CHAIN
        ORDER IS ALWAYS FIRST
        OWNER IS com-cli
        MEMBER IS ligne-cc MANDATORY AUTOMATIC
        SET SELECTION IS THRU CURRENT OF SET.

SET NAME IS ligne-prod
        MODE IS CHAIN
        ORDER IS ALWAYS FIRST
        OWNER IS produit
        MEMBER IS ligne-cc MANDATORY AUTOMATIC
        SET SELECTION IS THRU CURRENT OF SET.

SET NAME IS com-exp
        MODE IS CHAIN
        ORDER IS ALWAYS FIRST
        OWNER IS com-cli
        MEMBER IS expedition MANDATORY AUTOMATIC
        SET SELECTION IS THRU CURRENT OF SET.

SET NAME IS exp-lig
        MODE IS CHAIN
        ORDER IS ALWAYS FIRST
        OWNER IS expedition
        MEMBER IS ligne-exp MANDATORY AUTOMATIC
        SET SELECTION IS THRU CURRENT OF SET.
```

```
SET NAME IS com-exp-fus
        MODE IS CHAIN
        ORDER IS ALWAYS FIRST
        OWNER IS com-cli
        MEMBER IS exp-fus MANDATORY AUTOMATIC
        SET SELECTION IS THRU CURRENT OF SET.

SET NAME IS exp-fus-exp
        MODE IS CHAIN
        ORDER IS ALWAYS LAST
        OWNER IS exp-fus
        MEMBER IS expedition OPTIONAL MANUAL
        SET SELECTION IS THRU CURRENT OF SET.

SET NAME IS livr-ligne
        MODE IS CHAIN
        ORDER IS ALWAYS LAST
        OWNER IS livraison
        MEMBER IS ligne-livr MANDATORY AUTOMATIC
        SET SELECTION IS THRU CURRENT OF SET.


SUB-SCHEMA NAME IS ss-petitpas.

AREA SECTION.
        COPY ALL AREAS.

RECORD SECTION.
        COPY ALL RECORDS.

SET SECTION.
        COPY ALL SETS.

END-SCHEMA.
```

4. EXEMPLE DE GENERATION

4.1. LES COMMANDES DE GENERATION

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!                                                !!
!! E X E M P L E   D E   T E X T E   G E C O L 2 !!
!!                                                !!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!   Ceci est un exemple qui doit montrer, comment on
! peut generer du texte contenant de l'information
! d'une base de donnees. Dans le cas present la base de
! donnees est celle du cas PETITPAS. Le schema d'acces
! de cette base a ete donnee au debut de cette annexe.
!
@BEGIN
****************************************
*                                      *
* La base de donnees de P E T I T P A S *
*                                      *
****************************************

========================================
 La base de donnees
========================================

@FOR-EACH BD DO
 nom de la bd             : #BDIDEN
 mot de passe de la bd : #BDPASW
 nom de l'interface       : #BDINTE
@OD

========================================
 Les fichiers
========================================

@MOVE '0' TO #NUME01
@FOR-EACH FICHIER DO
@ADD 1 TO #NUME01 GIVING #NUME01
 nom du fichier           : #FIIDEN
@OD

 nombre de fichiers dans la base de donnees: #NUME01

========================================
 Les types d'articles
========================================

@MOVE '0' TO #NUME01
@FOR-EACH TARTICLE DO
@ADD 1 TO #NUME01 GIVING #NUME01
 nom du type d'article : #TAIDEN
@OD

 nombre de types d'articles dans la bd : #NUME01
```

```
==========================================
 Les types de chemins d'acces
==========================================

@MOVE '0' TO #NUME01
@FOR-EACH TCHEMIN DO
@ADD 1 TO #NUME01 GIVING #NUME01
 nom du type de chemin : #TCIDEN
@  FOR-EACH ORIGINE WITHIN TCOR DO
!
! acceder a ORIGINE par le chemin d'acces
! TCOR ( tchemin -> origine )
!
@    FOR-EACH TARTICLE USING #TACODE EQUAL #ORCOTA DO
!
! ORIGINE ne contient pas le nom du type d'article
! origine mais seulement le code de ce type d'article,
! qui est une cle identifiante. Pour retrouver le nom de
! du type origine on fait donc un acces par cle au type
! d'article .
!
                origine  : #TAIDEN
@    OD
@  OD
@  FOR-EACH CIBLE WITHIN TCCI DO
@    FOR-EACH TARTICLE USING #TACODE EQUAL #CICOTA DO
                cible    : #TAIDEN
@    OD
@  OD

@OD
 nombre de types de chemins dans la bd : #NUME01

==========================================
 Pour chaque type d'article les types de
 chemins dont il est origine
==========================================

@FOR-EACH TARTICLE DO
 type d'article : #TAIDEN
@ FOR-EACH TCHEMIN DO
@  FOR-EACH ORIGINE WITHIN TCOR DO
@   IF #TACODE = #ORCOTA THEN
       origine du chemin #TCIDEN
@   FI
@  OD
@ OD

@OD
```

```
==========================================
 Pour chaque type d'article les types de
 chemins dont il est cible
==========================================

@FOR-EACH TARTICLE DO
  type d'article  : #TAIDEN
@ FOR-EACH TCHEMIN DO
@  FOR-EACH CIBLE WITHIN TCCI DO
@   IF #TACODE = #CICOTA THEN
       cible du type de chemin : #TCIDEN
@   FI
@  OD
@ OD

@OD

==========================================
 Les items par type d'article
==========================================

@MOVE '0' TO #NUME01
@FOR-EACH TARTICLE DO
@ MOVE '0' TO #NUME02
  type d'article        : #TAIDEN
@ FOR-EACH ITEM WITHIN TAIT DO
@  ADD 1 TO #NUME01 GIVING #NUME01
@  ADD 1 TO #NUME02 GIVING #NUME02
           item         : #ITIDEN
@ OD
   #NUME02 items sont rattaches au type "#TAIDEN"

@OD


  il y a #NUME01 items dans la base de donnees

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!  Ceci termine l'exemple pour le langage  !!
!!  des macros : GECOL2                     !!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
@END
```

4.2. LE TEXTE GENERE

```
*****************************************
*                                       *
* LA BASE DE DONNEES DE P E T I T P A S *
*                                       *
*****************************************

=========================================
 LA BASE DE DONNEES
=========================================

 NOM DE LA BD           : SS-PETITPAS
 MOT DE PASSE DE LA BD  :
 NOM DE L'INTERFACE     : PTP1

=========================================
 LES FICHIERS
=========================================

 NOM DU FICHIER         : AR-CLIENT
 NOM DU FICHIER         : AR-COM
 NOM DU FICHIER         : AR-PRODUIT

 NOMBRE DE FICHIERS DANS LA BASE DE DONNEES: 0000000003

=========================================
 LES TYPES D'ARTICLES
=========================================

 NOM DU TYPE D'ARTICLE : CLIENT
 NOM DU TYPE D'ARTICLE : COM-CLI
 NOM DU TYPE D'ARTICLE : EXP-FUS
 NOM DU TYPE D'ARTICLE : EXPEDITION
 NOM DU TYPE D'ARTICLE : IDENT-RES-CLI
 NOM DU TYPE D'ARTICLE : LIBELLE-RESUME
 NOM DU TYPE D'ARTICLE : LIGNE-CC
 NOM DU TYPE D'ARTICLE : LIGNE-EXP
 NOM DU TYPE D'ARTICLE : LIGNE-LIVR
 NOM DU TYPE D'ARTICLE : LIVRAISON
 NOM DU TYPE D'ARTICLE : PRODUIT

 NOMBRE DE TYPES D'ARTICLES DANS LA BD : 0000000011

=========================================
 LES TYPES DE CHEMINS D'ACCES
=========================================

 NOM DU TYPE DE CHEMIN : COM-EXP
              ORIGINE  : COM-CLI
              CIBLE    : EXPEDITION

 NOM DU TYPE DE CHEMIN : COM-EXP-FUS
              ORIGINE  : COM-CLI
              CIBLE    : EXP-FUS
```

```
NOM DU TYPE DE CHEMIN : COM-LIGNE
            ORIGINE   : COM-CLI
            CIBLE     : LIGNE-CC

NOM DU TYPE DE CHEMIN : EXP-FUS-EXP
            ORIGINE   : EXP-FUS
            CIBLE     : EXPEDITION

NOM DU TYPE DE CHEMIN : EXP-LIG
            ORIGINE   : EXPEDITION
            CIBLE     : LIGNE-EXP

NOM DU TYPE DE CHEMIN : I-COM-EXP
            ORIGINE   : EXPEDITION
            CIBLE     : COM-CLI

NOM DU TYPE DE CHEMIN : I-COM-EXP-FUS
            ORIGINE   : EXP-FUS
            CIBLE     : COM-CLI

NOM DU TYPE DE CHEMIN : I-COM-LIGNE
            ORIGINE   : LIGNE-CC
            CIBLE     : COM-CLI

NOM DU TYPE DE CHEMIN : I-EXP-FUS-EXP
            ORIGINE   : EXPEDITION
            CIBLE     : EXP-FUS

NOM DU TYPE DE CHEMIN : I-EXP-LIG
            ORIGINE   : LIGNE-EXP
            CIBLE     : EXPEDITION

NOM DU TYPE DE CHEMIN : I-LIGNE-PROD
            ORIGINE   : LIGNE-CC
            CIBLE     : PRODUIT

NOM DU TYPE DE CHEMIN : I-LIVR-LIGNE
            ORIGINE   : LIGNE-LIVR
            CIBLE     : LIVRAISON

NOM DU TYPE DE CHEMIN : LIGNE-PROD
            ORIGINE   : PRODUIT
            CIBLE     : LIGNE-CC

NOM DU TYPE DE CHEMIN : LIVR-LIGNE
            ORIGINE   : LIVRAISON
            CIBLE     : LIGNE-LIVR

NOMBRE DE TYPES DE CHEMINS DANS LA BD : 0000000014
```

```
==========================================
 POUR CHAQUE TYPE D'ARTICLE LES TYPES DE
 CHEMINS DONT IL EST ORIGINE
==========================================

 TYPE D'ARTICLE : CLIENT

 TYPE D'ARTICLE : COM-CLI
       ORIGINE DU CHEMIN COM-EXP
       ORIGINE DU CHEMIN COM-EXP-FUS
       ORIGINE DU CHEMIN COM-LIGNE

 TYPE D'ARTICLE : EXP-FUS
       ORIGINE DU CHEMIN EXP-FUS-EXP
       ORIGINE DU CHEMIN I-COM-EXP-FUS

 TYPE D'ARTICLE : EXPEDITION
       ORIGINE DU CHEMIN EXP-LIG
       ORIGINE DU CHEMIN I-COM-EXP
       ORIGINE DU CHEMIN I-EXP-FUS-EXP

 TYPE D'ARTICLE : IDENT-RES-CLI

 TYPE D'ARTICLE : LIBELLE-RESUME

 TYPE D'ARTICLE : LIGNE-CC
       ORIGINE DU CHEMIN I-COM-LIGNE
       ORIGINE DU CHEMIN I-LIGNE-PROD

 TYPE D'ARTICLE : LIGNE-EXP
       ORIGINE DU CHEMIN I-EXP-LIG

 TYPE D'ARTICLE : LIGNE-LIVR
       ORIGINE DU CHEMIN I-LIVR-LIGNE

 TYPE D'ARTICLE : LIVRAISON
       ORIGINE DU CHEMIN LIVR-LIGNE

 TYPE D'ARTICLE : PRODUIT
       ORIGINE DU CHEMIN LIGNE-PROD


==========================================
 POUR CHAQUE TYPE D'ARTICLE LES TYPES DE
 CHEMINS DONT IL EST CIBLE
==========================================

  TYPE D'ARTICLE  : CLIENT

  TYPE D'ARTICLE  : COM-CLI
       CIBLE DU TYPE DE CHEMIN : I-COM-EXP
       CIBLE DU TYPE DE CHEMIN : I-COM-EXP-FUS
       CIBLE DU TYPE DE CHEMIN : I-COM-LIGNE

  TYPE D'ARTICLE  : EXP-FUS
       CIBLE DU TYPE DE CHEMIN : COM-EXP-FUS
       CIBLE DU TYPE DE CHEMIN : I-EXP-FUS-EXP
```

```
  TYPE D'ARTICLE  : EXPEDITION
       CIBLE DU TYPE DE CHEMIN : COM-EXP
       CIBLE DU TYPE DE CHEMIN : EXP-FUS-EXP
       CIBLE DU TYPE DE CHEMIN : I-EXP-LIG

  TYPE D'ARTICLE  : IDENT-RES-CLI

  TYPE D'ARTICLE  : LIBELLE-RESUME

  TYPE D'ARTICLE  : LIGNE-CC
       CIBLE DU TYPE DE CHEMIN : COM-LIGNE
       CIBLE DU TYPE DE CHEMIN : LIGNE-PROD

  TYPE D'ARTICLE  : LIGNE-EXP
       CIBLE DU TYPE DE CHEMIN : EXP-LIG

  TYPE D'ARTICLE  : LIGNE-LIVR
       CIBLE DU TYPE DE CHEMIN : LIVR-LIGNE

  TYPE D'ARTICLE  : LIVRAISON
       CIBLE DU TYPE DE CHEMIN : I-LIVR-LIGNE

  TYPE D'ARTICLE  : PRODUIT
       CIBLE DU TYPE DE CHEMIN : I-LIGNE-PROD


==========================================
 LES ITEMS PAR TYPE D'ARTICLE
==========================================

  TYPE D'ARTICLE       : CLIENT
           ITEM        : NUM-CLI
           ITEM        : NOM-CLI
           ITEM        : PRE-CLI
           ITEM        : ADR-CLI
   0000000004 ITEMS SONT RATTACHES AU TYPE "CLIENT"

  TYPE D'ARTICLE       : COM-CLI
           ITEM        : NUM-CC
           ITEM        : NUM-CLICC
           ITEM        : DATE-CC
           ITEM        : ETAT-CC
   0000000004 ITEMS SONT RATTACHES AU TYPE "COM-CLI"

  TYPE D'ARTICLE       : EXP-FUS
           ITEM        : DATE-EXP-FUS
           ITEM        : NUM-EXP
   0000000002 ITEMS SONT RATTACHES AU TYPE "EXP-FUS"

  TYPE D'ARTICLE       : EXPEDITION
           ITEM        : DATE-EXP
           ITEM        : NUM-FACTURE
   0000000002 ITEMS SONT RATTACHES AU TYPE "EXPEDITION"
```

```
TYPE D'ARTICLE        : IDENT-RES-CLI
          ITEM        : NOM-RES-CLI
          ITEM        : PRE-RES-CLI
          ITEM        : ADR-RES-CLI
 0000000003 ITEMS SONT RATTACHES AU TYPE "IDENT-RES-CLI"

TYPE D'ARTICLE        : LIBELLE-RESUME
          ITEM        : LIB-RES
 0000000001 ITEMS SONT RATTACHES AU TYPE "LIBELLE-RESUME"

TYPE D'ARTICLE        : LIGNE-CC
          ITEM        : Q-COM-C
          ITEM        : Q-DUE-C
          ITEM        : NUM-PRO-CC
 0000000003 ITEMS SONT RATTACHES AU TYPE "LIGNE-CC"

TYPE D'ARTICLE        : LIGNE-EXP
          ITEM        : Q-LIVR-C
          ITEM        : NUM-PRO-EXP
 0000000002 ITEMS SONT RATTACHES AU TYPE "LIGNE-EXP"

TYPE D'ARTICLE        : LIGNE-LIVR
          ITEM        : NUM-PROD
          ITEM        : QUANTITE
 0000000002 ITEMS SONT RATTACHES AU TYPE "LIGNE-LIVR"

TYPE D'ARTICLE        : LIVRAISON
          ITEM        : NUM-LIVR
 0000000001 ITEMS SONT RATTACHES AU TYPE "LIVRAISON"

TYPE D'ARTICLE        : PRODUIT
          ITEM        : NUM-PRO
          ITEM        : LIBELLE-PRO
          ITEM        : Q-EN-STOCK
          ITEM        : Q-DIFFEREE
          ITEM        : POIDS-U
          ITEM        : P-UNIT
          ITEM        : I-EPUIS
          ITEM        : U-MES
          ITEM        : Q-PAR-DEFAUT
 0000000009 ITEMS SONT RATTACHES AU TYPE "PRODUIT"


IL Y A 0000000033 ITEMS DANS LA BASE DE DONNEES
```

LE GUIDE DES CHANGEMENTS DU SYSTEME

août 1982

TABLE DES MATIERES

1. INTRODUCTION

Probablement il s'avèrera nécessaire d'apporter tôt ou tard certaines modifications au système de génération, et ceci pour deux raisons principalement.

D'une part un des objectifs que nous avons poursuivi lors de la réalisation de ce système a été d'aboutir a un système pouvant faire face à des exigences diverses. Mais comme ces exigences n'ont pas toutes pu être prévues il sera peut-être nécessaire d'adopter le système à de nouvelles exigences, comme par exemple accepter une nouvelle directive etc.

D'autre part la complexité du système, sa taille et certaines contraintes de temps nous ont amenés à procéder à des choix parfois arbitraires qui pourraient se révéler être malheureux lors de l'exploitation du système.

Consients de ces problèmes nous avons essaié d'aboutir à un système qui peut être modifié facilement dans certains cas.

Rappellons que le système a été réalisé dans le but de permettre la génération d'interfaces d'accès entre le modèle d'accès généralisé et toutes les bases de données sur tous les SGBD où il est possible d'établir la correspondance entre le MAG et le modèles du SGBD. Par conséquent il faut voir les changements dans deux classes différentes.

D'un côté il y a les changements ou extensions "normaux" car le la version courante du système a été réalisée dans le but d'accepter ces changements. Ce sont:

- changement d'un SGBD existant déjà dans le système (DBMS-20)

- extension du système à un nouveau SGBD

D'un autre côté il y a les changements pour lesquels le système n'a pas été prévu parce qu'on s'est basé sur la fixité de ces éléments. Il est évident que ces changements sont en général beaucoup plus complexes à réaliser. Ces changements sont dans un nombre important.C'est pourquoi nous nous sommes limités ici à ceux qui nous semblent raisonnablement envisageables. Ce sont:

- certaines modifications du MAG

- extensions du langage GECOL2

2. CHANGEMENT D'UN SGBD EXISTANT DANS LE SYSTEME

L'apparition d'une nouvelle version d'un SGBD existant dans le système de génération peut entraîner des modifications au niveau de l'analyseur et des macros liés à ce SGBD.

Une modification dans le DDL entraîne vraisemblablement un changement au niveau de l'analyseur. Il faut repérer dans le programme de l'analyseur l'endroit qui traite la partie du DDL changée et adapter cette partie en fonction. S'il y a une ajoute au DDL, il faut introduire une nouvelle partie dans l'analyseur prenant en charge cette ajoute. Pour pouvoir faire ces adaptations pour DBMS-20 on s'appuyera sur les documentations suivantes:

1. le chapitre sept du mémoire qui explique le fonctionnement de l'analyseur

2. le manuel d'utilisation de l'analyseur qui précise les restrictions observées dans la version courante de l'analyseur

3. le manuel d'utilsation de la méta-bd (partie "création") où sont données les informations nécessaires à la création d'un méta-article dans la base de données des schémas

Il faut s'assurer en plus que après ces changements suite à une modification du DDL tous les informations utilisées dans les macros sont toujours chargées dans la base de données des schémas. Sinon il faut faire les changements nécessaires comme décrits dans la partie ci-dessous.

En général un changement de DML entraîne une modification des macros. Pour faire ces modifications il faut repérer toutes les fonctions du MAG qui sont affectées par ce changement et par la suite procéder à la modification des parties dans les macros liées a ces fonctions. Pour faire les modification on se basera sur:

1. le manuel de l'utilisateur du SGBD car ceci suppose qu'on ait une connaissance suffisante du SGBD en question

2. le mauel d'utilisation du générateur où il est expliqué d'une part comment écrire des macros et d'autre part comment il faut introduire de nouvelles macros dans le système sur le système d'exploitation courant: TOPS-20

3. connaissance du langage lié au MAG: consulter le manuel d'utilisation des interfaces

3. INTRODUCTION D'UN NOUVEAU SGBD

L'extension à un nouveau SGBD nécessite d'une part un nouveau analyseur et d' autre part un nouveau texte de génération. Pour pouvoir procéder à l'implémentation de ceux-ci il faut avoir réalisé la correspondance entre les modèles de ce SGBD et le MAG.

Si l'on écrit l'analyseur on peut procéder comme l'on veut à conditiond'assurer que la méta-base soit bien garnie avec toutes les informations dans le format exigé. Ces exigences du point de vue de la méta-base de données sont formulées dans le manuel d'utilisation de la méta-bd. Pour plus de détail on peut consulter le chapitre quatre du mémoire.

Pour écrire les macros il faut avoir d'une part une connaissance suffisante du langage du modèle d'accès; information qui se trouve dans le manuel d'utilisation des interfaces. D'autre part il faut connaître le DML du SGBD. Enfin il faut connaître le langage de génération GECOL2 et la manière d'utiliser le générateur, ce qui est décrit dans le manuel d'utilisation du générateur.

4. MODIFICATION DU MAG

Ceci est une modification qui n'a pas été prévue lors de la conception du système. C'est pourquoi il est difficile de prendre en charge une telle modification. Nous nous limiterons ici à deux modifications:

- introduction d'un nouveau méta-item

- introduction d'une nouvelle méta-clé

Ces modifications-ci nécessitent des changements dans les programmes GENER et METABD. Pour pouvoir faire cela, il faut avoir une connaissance suffisante de COBOL.

Dans la partie qui va suivre nous ne verrons que les modifications nécessaires aux programmes. Cependant il est conseillé vivement de faire en même temps la mise-à-jour de la documentation, donc principalement les manuels d'utilisation.

4.1. NOUVEAU META-ITEM

On suppose que l'on veut ajouter un nouveau méta-item MI au type de méta-article TMA. Pour ne pas trop compliquer, ajouter toujours à la fin du méta-article. Il faut remarquer que le nom à utiliser pour le méta-item doit être constitué de six caractères. En plus ce nom doit être choisi de telle manière qu'il identifie le méta-item parmi tous les méta-items de la base de données.

Dans la méta-base de données:

1. Dans "tables des objets" agrandir le FILLER dans la table correspondant à TMA de la longueuer de MI. Si maintenant la longueur de la ligne du tableau dépasse 93 il faut allonger FOBJ-TEXT dans le fichier des méta-objets de la longueur correspondante. Changer dans ce dernier cas aussi la description de ces fichiers dans tous les programmes d'analyse. En plus il faut adapter la longueur de RFIELD dans la LINKAGE SECTION de METABD .

Dans le générateur:

1. dans la "table de gestion des boucles" ajouter le nom du méta-item en six caractères dans la redéfinition de FE-record qui correspond au TMA. Il faut prendre la place sur le FILLER s'il y en a assez, sinon il faut étendre la longueur de FE-RECORD et de toutes les redéfinitions.

2. insérer le nom du MI dans "table des données" selon l'ordre alphabétique. Allonger la taille de cette table et augmenter la valeur du maximum MAX-DATA-TABLE.

3. si l'on a fait un allongement de RFIELD dans METABD, faire le même allongement dans RFIELD dans "paramètres de l'appel bd-schémas"

4. dans DEBUT-FIND-PARAM-VALUE de COMMON SECTION dans le GO TO ... DEPENDING insérer un nouveau branchement selon l'ordre alphabétique à un nouveau paragraphe; rédiger ce paragraphe en prenant l'exemple des autres paragraphes Wxxxx

4.2. NOUVELLE META-CLE

Supposons qu'on veut introduire une nouvelle méta-clé d'accès MC pour le type de méta-article TMA. On suppose que les méta-items qui composent MC existe déjà. Pour ne pas trop compliquer on admet pour cette clé seulement une recherche séquentielle.

La méta-clé est identifiée parmi toutes les autres méta-clés de la méta-bd par son nom qui a au maximum six caractères.

Dans la méta-bd:

1. dans la section TRAITEMENT34 SECTION "Accès par clé aux articles d'un type dans un fichier":

2. dans le GO TO ... DEPENDING ajouter TR34(nn+1) après TR34nn, si celui-ci a été le dernier jusqú à présent

3. créer les paragraphes TR34(nn+1) et TR43(nn+1)-LOOP en prenant comme modèle TR3401 et TR3401-LOOP; il faut cependant remarquer que ceci est le traitement pour une clé identifiante; si la nouvelle clé MC ne l'est pas il convient de faire les changements nécessaires

Dans le générateur:

1. ajouter la clé dans "table des clé d'accès" à la fin et allonger la taille de la table; modifier aussi la valeur du nombre maximum de clés MAX-KEY

5. EXTENSIONS DU LANGAGE DE GENENRATION

Une extension du langage de gênêration a seulement un effet au niveau du gênêrateur. Il est assez facile d'introduire une nouvelle directive dans GECOL2 et de la faire accepter par le gênêrateur.

Une directive est identifiée par son premier mot. Il faut donc choisir tout d'abord un mot identifiant pour la directive et de faire débuter celle-ci par ce mot. Les arguments pouvant intervenir dans une directive sont ceux définis dans 2.7. à 2.10. du manuel d'utilisation du gênêrateur.

Dans le programme même il suffit d'ajouter une branche en plus dans l'aiguillage se faisant sur le mot identifiant de la directive dans le paragraphe TRAIT-DIRECTIVE de TRAITEMENT-DIRECTIVE SECTION. Chaque directive a une section qui lui est propre. Il s'agit de crêer cette section qui termine l'analyse de la directive à partir du deuxième mot de celle-ci et qui se charge de l'exécuter.